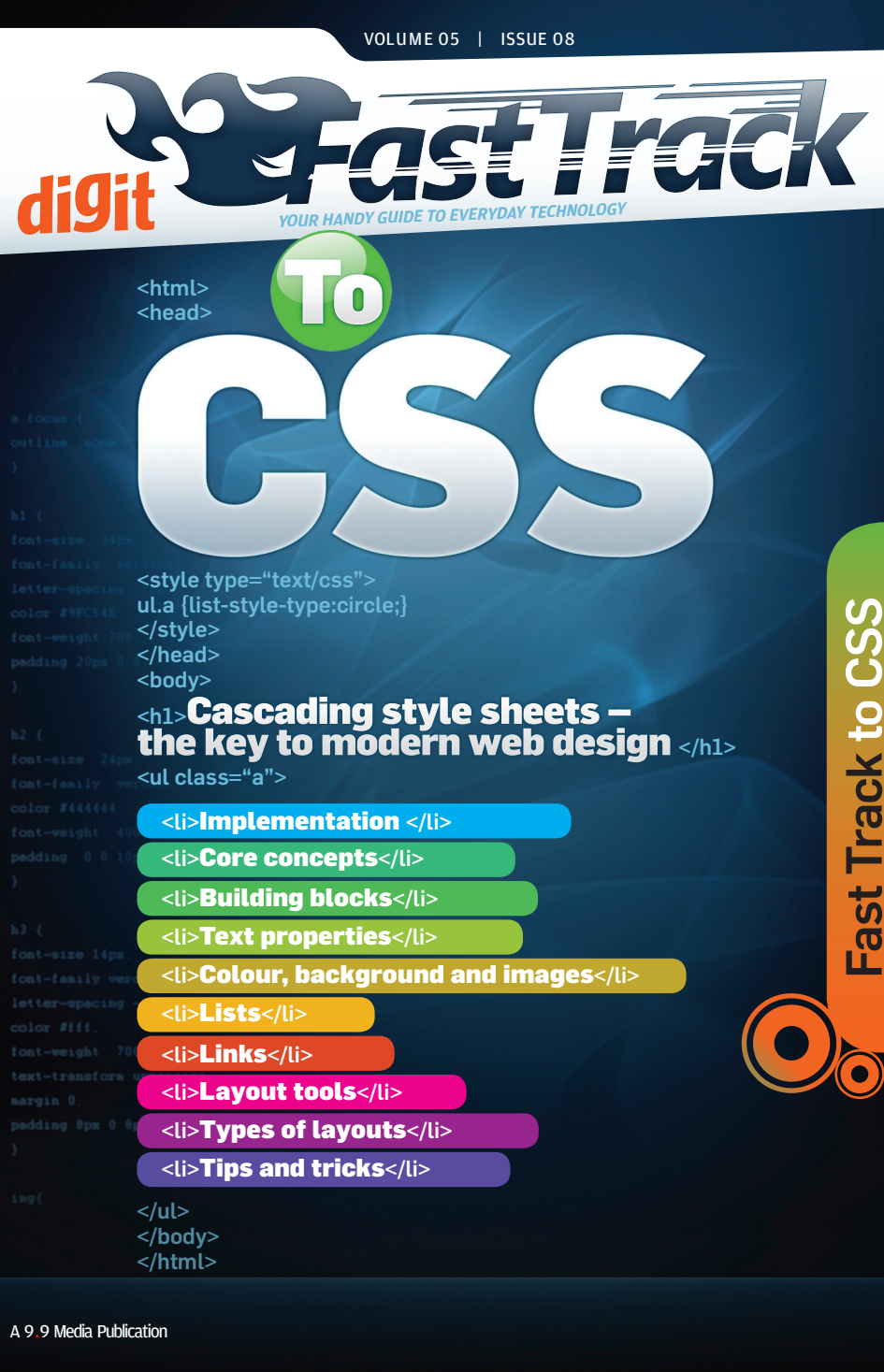
VOLUME 05 | ISSUE 08

digit Fast Track
YOUR HANDY GUIDE TO EVERYDAY TECHNOLOGY



Fast Track to CSS

A 9.9 Media Publication

digit
YOUR TECHNOLOGY NAVIGATOR
thinkdigit.com

Fast Track to CSS

thinkdigit.com

Credits

The People Behind This Book

Editorial

Editor	Robert Sovereign-Smith
Head-Copy Desk	Nash David
Writer	Hanu Prateek K

Design and Layout

Lead Designer	Vijay Padaya
Senior Designer	NV Baiju
Cover Design	Anil T

AUGUST 2010

Free with Digit. Not to be sold separately. If you have paid separately for this book, please email the editor at editor@thinkdigit.com along with details of location of purchase, for appropriate action.

Contents

Introduction

From the early years of the internet, HTML has always been the fundamental format for transmitting information over the web. HTML which stands for HyperText Markup Language was the primary language used to develop webpages over the internet. But initially, HTML started off as a purely structural markup language composed of structural elements to define things like paragraphs, hyperlinks, lists, and headings in text documents.

But as computers and the internet evolved, visual components started getting incorporated into the web and HTML. With this increased interactivity and the appearance of colour displays, aesthetics and design started becoming very important. More and more was required from HTML in terms of presentation. Subsequently HTML had to evolve and incorporate components such as `<font>` and `<big>` to take care of the presentational requirements. But soon even this wasn't enough, even more presentation control was demanded from HTML. As a result tables, frames and other complex markup were incorporated into HTML to help with the presentation of web pages. As the internet grew, web pages became complex and suddenly, the structural language became presentational.

To find a solution to this problem and establish standards across the internet, style sheet languages were proposed by the W3C (World Wide Web Consortium). CSS along with new HTML standards termed XHTML was thus developed to separate the presentation aspect of web design from HTML and restrict it to the structural markup and content only. CSS would take care of the presentational requirements. There were other major advantages of following this approach, which included reduced code, more accessibility and flexibility, lighter web pages and the ability to make site-wide changes across multiple HTML files in an instant.

CSS which stands for Cascading Style Sheets, is the language used to define the presentational aspect of web pages. One of the major advantages of CSS is its simple and easy to understand syntax. Here the code is written to define the presentation and the styling of elements present in a document written with a markup language like HTML or XML. For the rest of this book we will be looking at CSS specifically in conjunction with HTML.

It is important to understand that CSS is designed specifically to separate textual content from its presentation. This crucial division of structure and presentation helps us in getting results faster and creating more compelling

content. The separation of structure and presentation gives us the ability to change the appearance and presentation of content without having to touch the content.

For the most part, any strange references in this book will have been explained then and there or somewhere in the prior chapters. If you follow the chapters in the exact sequence as presented, there shouldn't be a problem. Also understand that we are going ahead with the assumption that you have some basic elementary understanding of HTML code. If you do not, HTML is a very simple language and it is easy to follow along after you have referred to the many online resources available for HTML.

CSS is a tricky language, primarily because of the fact that there is a difference in how different browsers render it. Older versions of browsers like Internet Explorer do not necessarily follow the correct CSS standards. The code in this book is standards complaint and should be rendered just as expected in all major modern browsers. It is assumed that in all examples containing HTML code, the code is placed within the `<body>` tag of a valid HTML document, while the CSS is placed in an external style sheet linked from the `<head>` of the HTML document.

Also note that for writing your code you can use software like Dreamweaver and Netbeans for both HTML and CSS, and in case you do not have them, the simple notepad will do. But word processing software like Microsoft Word should not be used.

1 CSS Implementation

No matter how well designed and well written the CSS code might be, it is of no use on its own. The CSS styles have to be applied to a clearly structured HTML web page. It is important to understand how the implementation of CSS works. Let's look at the different ways of applying CSS and their benefits and drawbacks.

1.1 The Basic HTML template

For all the examples in this chapter we will be using a simple HTML template having some standard document sections. Keep in mind that we are moving ahead with the assumption that you have some elementary understanding of HTML code.

To really understand and comprehend CSS, it's recommended that you copy the code from this template verbatim and try out all the examples yourself.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//
EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
  <title>CSS Implementation</title>
</head>
<body>
  <h1> CSS Implementation </h1>
  <p>This is a test website containing several HTML
  templates, each being styled using a different method
  of CSS application.</p>
  <p>Click an example below.</p>
  <h2>Examples</h2>
  <ul>
      <li><a href="base.html">Base template</a></li>
      <li><a href="inline.html">Inline CSS</a></li>
      <li><a href="embedded.html">Embedded CSS</a></li>
      <li><a href="external.html">External CSS</a></li>
      <li><a href="imported.html">Imported CSS</a></li>
      </ul>
```

```
</body>
</html>
```

1. Begin by creating a new file called base.html and opening it with a text editor such as Notepad or Wordpad (do not use word processors such as MS Word).
2. Add the HTML code above, to it.
3. Save the file to a new folder on your computer.
4. Open the file using a browser to see the basic web page as it stands.
5. Now let's make more template pages, each of which we will be using to show different ways in which CSS can be implemented. Copy base.html and make four files and name them inline.html, embedded.html, external.html, and imported.html.
6. Save these new files to the same folder as base.html.
7. The four new files should now be available from your base.html file in your web browser.

1.2 Inline styles

The first method of implementing CSS is through inline styles. This is the most basic way of doing it, making use of the style attribute for specific tags within the HTML document itself. The styles are declared using the format `property:value`. We will go into the CSS syntax and the different properties and their values a little later. To try this out,

1. Open inline.html in your text editor.
2. Proceed to the first opening paragraph tag <p>.
3. Replace the <p> with `<p style="color: red">` and save the template.

Save and reload the file in your browser. Notice now that the text contained within that paragraph will be red. Only that particular paragraph is affected, and the second paragraph defaults to black. This method of applying styles can be applied to any HTML element within the <body> of the page.

Inline styles are useful for testing out simple CSS rules instantly, however, as you get more proficient with CSS you will realize that it is not the best way of doing things. The underlying of CSS itself is keeping the HTML free from the code required for presentation. Using inline styles will increase your individual file size significantly and also requires you to declare your styles for each and every tag individually. Imagine having to style every individual section of a website containing hundreds of pages.

1.3 Embedded styles

The idea behind embedded styles is that you still have your styles in the same HTML document, but grouped together in the head of the document as a single element which is gets applied page wide.

Let's try it out,

1. Open embedded.html in your text editor.
2. Within the `<head>` section of the template, just after the `<title>` element, paste the following code: `<style type="text/css">p {color: red;}</style>`
3. Again, we are using a simple CSS declaration to render the text red.
4. Save the template, and open it in the web browser.

Notice now that the text contained within both the paragraphs will be red. This time around, all paragraphs in the document are affected by this declaration, as the style is applied to all `<p>` tags within the page.

This method is definitely more effective than inline CSS and it allows you to administer styles throughout the document. But grouping the styles together within the HTML document still increases your file size. The problem with embedded styles is again the fact that they are loading from within the document. Also, these styles will be downloaded again and again every time with every page load, every page of your web site will need its own embedded styles, and making site-wide style changes will be very laborious.

1.4 External styles

The most commonly used method for implementing CSS and also the most effective is making use of external style sheets.

Again let us work with an example

1. Open external.html in your text editor.
2. Within the `<head>` section, and after the `<title>` element, paste the element `<link rel="stylesheet" type="text/css" href="external.css" />` and save the file. This line tells the browser to look for an external file called `external.css`, which is a CSS file, and is stored in the same directory as the HTML file.
3. Now create a new file called `external.css`.
4. Paste the following code `p {color: red;}` into `external.css`.
5. Save external.css to the same folder as the .html files, and again open it in your web browser.

Again the paragraphs turn red, but note that there is not a single CSS rule or style element anywhere in your HTML file. Your HTML is free from

the presentation code and the colour is being controlled via the external style sheet. From this point on whatever changes are needed within the site in terms of style and presentation can be done directly through a single external style sheet. Also, once a browser accesses the style sheet, it caches it and wouldn't need to keep downloading it each time it accesses it. Rather, it only downloads the page content. This will make your website faster and also keep your HTML clean.

1.5 Importing styles

The `@import` rule is a method for importing one or more style sheets via the main external style sheet.

Continuing from the previous example,

1. Open the `imported.html` file in your text editor.
2. Within the `<head>` section, and after the `<title>` element, paste the element `<style type="text/css">@import url(external.css);</style>`
3. Save `imported.html` and view it in the browser. Nothing looks different, and the paragraphs are still red, but this is a sensible and productive way of style sheet management.

The `@import` rule is also useful for hiding your style sheet from really old out-dated browsers with poor CSS support such as Netscape 4.x, and IE4 that don't support `@import`. Rather than attempt to do something half-useful with your CSS, these browsers will just completely ignore your style sheet and leave the HTML unstyled.

1.6 Maintaining and organising style sheets

1.6.1 Style Sheets for different media

All modern browsers support the common media attributes that are applied within the `<link>` element to target specific style sheets in a specific situation. For example, to ensure only visitors viewing the web site on a monitor see your glamorous design, you add `media="screen"` to the `<link>` element to call your default style sheet. Underneath that, a second `<link>` element can be used with `media="print"` added to call a print style sheet with only basic styling such as black text on a plain white background, and all graphics removed:

```
<link rel="stylesheet" media="screen" type="text/css"
href="screen.css" />
```

```
<link rel="stylesheet" media="print" type="text/css"
href="print.css" />
```

If a style sheet has a media type of screen, it will not be used when the page is printed. If no media type were specified, the style sheet would influence the printed result. Note also that any style sheet intended only for printing purposes must be given the print media type to prevent it from being implemented on screen. It is therefore very important to specify the media type correctly.

The next most popular media attribute is `media="handheld"`, which, while not supported by all mobile devices, is still in common use as browsers will ignore its content, allowing many cell phone or PDA users to access a stripped-down version of your styling depending upon support. Again, all that is required is another `<link>` element with `media="handheld"` specified, calling a specific style sheet such as `handheld.css` or `mobile.css`.

1.6.2 Multiple style sheets and the use of directories

As you keep increasing the number of rules used, your external style sheet will become really long and unmanageable. It is always recommended that you spilt your CSS into smaller chunks. Also, you might have separate style sheets for browsers, for print and for mobile devices if necessary. Soon you'll be looking at multiple style sheets for other reasons. The best approach to managing these is to use separate folders for each.

1.6.3 Modular CSS

An effective way of managing your CSS is dividing your CSS into modules, such as default key rules (font control, colour, headings) which apply to every page and some rules that will be applied to `<form>` elements used only on a few pages. You can create a specific style sheet for each module. For example, say a default module, forms module, navigation module, and so on. By combining each module, you will end up with the CSS for a complete site, yet everything remains separated, and you can navigate through them with ease.

In the following example, the basic style sheet (`external.css`) is used to import two modular ones (`default.css` and `layout.css`) to have a combined effect on the HTML:

1. Create a new style sheet called `default.css`.
2. Place the CSS `p {color: #red;}` in it and save the file.

3. Create another style sheet called `layout.css`.
4. Place the following CSS into `layout.css`: `#header {height: 100px; width: 100%; border: 1px solid #999;};` and save the file.
5. Open `external.css` and replace the existing CSS with the following:

```
@import url("default.css");
@import url("layout.css");
```

6. Save the file.
7. Open external.html and ensure the link to external.css is still in place.
8. Find `<h1>Applying CSS Templates</h1>` and wrap this with the hooks for the header, so you end up with `<div id="header"><h1>Applying CSS Templates</h1></div>`
9. Save `external.html` and view it in your browser.

Notice that the CSS from `default.css` is still giving you a red paragraph, but also that the header rule from layout.css is placing a box of 100 pixels height by 100 pixels width around the heading text. The `external.css` file is successfully combining the two style sheets it imports for a complete effect.

1.6.4 Commenting

Defining rules is just the beginning. Consider how unmanageable a style sheet can become once it holds 20 or 30 rules. This is where commenting becomes invaluable. The following example includes simple comments that remind us what the rules are there for:

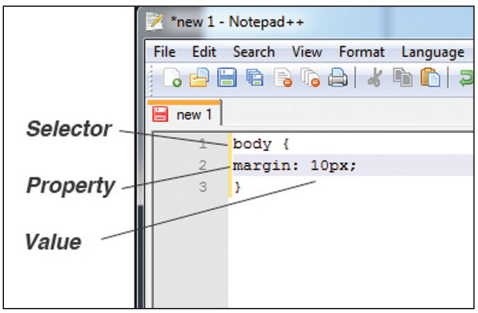
```
/* Default styling for paragraphs */
p {
color: #F00;
font-size: 12px;
}
/* Make all top-level headings gray and 16px high */
h1 {
color: #333;
font-size: 16px;
}
```

Introducing comments into style sheet makes things easier for any changes you would need to do later or if someone other than you has to work with the same code. It explains what your rule is supposed to be doing and is a good practice. All comments begin with a forward slash and asterisk (/*), and end with the asterisk followed by the forward slash (*/).

1.7 CSS syntax

Even though there are many alternatives to how CSS can be written compared to what we will see now, this is the most sensible way of structuring your CSS code. It will aid in navigation and debugging, and certainly make it easier for you or anyone else to revisit the CSS to make tweaks later on.

```
p {
color: red;
}
```



The correct CSS syntax

Rule 1: After the selector, the property and value are contained within curly braces.
Rule 2: A semi-colon always follows the declaration.
Rule 3: In the declaration, the Property (color) is followed by a colon, followed by a value (red).

Failing to include the rules in curly braces or missing the colon or semi colon will result in errors. Additional properties and values for the selector can be added within the curly braces.

```
p {
color: red;
font-size: 12px;
}
```

Now all the paragraphs will be red and have a set font size of 12 pixels. The selector (in this case the p) acts as the link between the CSS and the HTML, and as a result all paragraphs will be styled accordingly. Note that the selector is defined in lowercase, as required by XHTML only, as HTML is case insensitive.

For now, we'll concentrate on how a style sheet is structured. The basic templates you have created in this chapter will be useful as a starting point for other future examples.

2 Core concepts of CSS

In this chapter we will take a closer look at the core concepts of basic CSS design. These concepts are required to understand how CSS works, the intrinsic working of styling and what happens behind the scenes when applying these styles. We will be looking into concepts like selectors, cascade, grouping, inheritance and measurements

2.1 Selectors (ID and Class)

In the previous examples we have only seen a few base selectors in use for CSS. A base selector is basically an existing HTML tag and using CSS you can redefine some of its default properties to style the whole element. For example, we have already seen how CSS changes the text colour of a paragraph using the selector 'p' which is a basic HTML tag.

In addition to these selectors, CSS allows you define your own custom selectors, namely the ID and class selectors. IDs and classes are applied to HTML elements as attributes providing more control over the presentation. It is common among novice developers to confuse ID with class. But understanding the differences will make your work a lot easier.

2.1.1 IDs

An ID is a unique identifier to an element and can only be used once per page. Typically, an ID is used for any unique page element such as the header, main navigation, footer, or other key parts in the page design.

The ID is applied to an HTML element by referencing it in the HTML using the id="name" attribute immediately after the opening tag within an element. To illustrate this let us work with two IDs named highlight and default, respectively, and by applying them to two paragraphs:

```
<p id="highlight">This paragraph has red text.</p>
<p id="default">This paragraph has dark gray text.</p>
```

The corresponding CSS uses the hash (#) character to denote the rule using an ID as a selector.

```
/* Define highlighted text */
#highlight {
color:#red;
}
```

```
/* Define default text */
#default {
color:#333;
}
```

Existing or new IDs can be combined with selectors in the style sheet to add further control. Let's say only the main <h2> on your page needs to be emphasized with a different color. This calls for a new rule where the selector is defined in the form element#name:

```
/* Adjust the color of h2 when used as a title */
h2#title {
color:#red;
}
```

Here the new rule will override the default <h2> colour with red (color: #red;) whenever an <h2> is identified with the id="title" in the HTML. Simply add the unique identifier to the page:

```
<h2 id="title">Title Of My Article</h2>
```

IDs should be reserved for unique, single-use elements such as the header, sidebar, the main navigation or the page footer. This lets you scan through your markup easily, as all ID attributes will denote unique content areas or special regions of the page. They also provide greater flexibility for a more complex CSS application. IDs must be avoided when there is more than one requirement for the same CSS rule. Do not use an ID for anything you will need to duplicate in the future, such as images, link styles, or paragraphs.

2.1.2 Class

A class selector can be employed any number of times on the same page, making it a very flexible method for applying CSS. A class lets an element belong to a group where common style rules are applied, or itself is a reusable object or style. The most common way of applying a class selector just like an ID selector is referencing it in the HTML using a class="name" attribute of an HTML element.

As with our ID example, the two classes are named highlight (for red text) and default (for dark gray text):

```
<p class="highlight">This paragraph has red text.</p>
<p class="default">This paragraph has dark gray text.</p>
```

```
<p class="default">This also has dark gray text.</p>
```

Note that as the identifiers are classes, they can be used more than once, hence in this example two paragraphs have been identified as default. That would not be acceptable if IDs are used. The corresponding CSS uses a full stop (.) character to denote the rule is a reusable class. The full stop is combined with the class name to start the rule, followed by the property declarations:

```
/* Define highlight class */
.highlight {
color:#F00;
}
/* Define default class */
.default {
color:#333;
}
```

Classes are very useful when you want to have control over a number of elements. Consider the following drinks list.

```
<ul id="drinks">
<li class="alcohol">Beer</li>
<li class="alcohol">Spirits</li>
<li class="mixer">Cola</li>
<li class="mixer">Lemonade</li>
<li class="hot">Tea</li>
<li class="hot">Coffee</li>
</ul>
```

Note first that the unordered list (<ul>) is given a unique ID. Thus, id="drinks" will not be used again on the page at any time, allowing that particular list to be styled uniquely. Note also that Beer and Spirits are within list elements defined with class="alcohol", Cola and Lemonade are within list elements defined with class="mixer", and finally Tea and Coffee are defined in list elements with class="hot". This allows each drinks group to be treated individually. The CSS declares that the default text for that list will be red, so any list items without a class attribute will default to red text:

```
/* Drinks list styling */
ul#drinks {
```

```
color:#F00;
}


/* Define alcohol color */
.alcohol {
color:#333;
}

/* Define mixer color */
.mixer {
color:#999;
}

/* Define hot drinks color */
.hot {
color:#CCC;
}
```

The result sees the list of items move through shades of grey (defined by the classes). Any further drinks added to the list can be assigned to a particular drinks group, such as `<li class="alcohol">Wine</li>`

Use classes to control elements that belong to a group, for temporary items, and also to enhance the behaviour of an ID. It is recommended that classes are not used for styling structural elements within a page, such as the headers, main navigation, etc. Although it will work, doing that would reduce the flexibility of your design and make further customization difficult without any additional markup. Also, make sure if the class is actually necessary, and that the element cannot be targeted by defining a simple rule before defining it with a class. Remember that a class is used for exceptions to normal styling, and not to define the standard for an element.

2.2 The cascade

CSS stands for Cascading Style Sheets. So far we have seen the style and sheets part, but what does cascade mean? We just stated that a class value will override a base CSS rule. Well, there is an actual hierarchy in place according to which the rules are applied when there are overlapping rules. This is exactly what cascade implies. We have already seen different methods for applying CSS inline, embedded, external, and also importing. If you're storing your CSS

rules in an external style sheet that is dictating the presentation across your web site, and for some reason, you need to overrule some of the external styles for just one of your pages, you use cascading. For that one particular web page, you could use the embedded CSS in the `<head>` of the page, redefining the appropriate rules there itself. When this particular web page is loaded in the browser, it will apply the embedded CSS instead of applying your external CSS for the defined rules. Any identical selectors in the external style sheet will be ignored. Similarly one step up in the hierarchy is the inline styles. Whatever styles you apply inline will overrule any declarations in the `<head>` of the page or an external style sheet.

To see this in action, you can run through the following example:

1. Open external.css and define the default paragraph colour with `p {color:#red;}` and save the file.
2. View `external.html` in your browser. Assuming you are still applying CSS using `external.css`, any default paragraphs should be red.
3. Now open external.html and apply the embedded style `<style type="text/css">p {color: #333;}</style>` in the head of the template.
4. Reload `external.html` in your browser. Now any default paragraphs should be dark gray, as the embedded CSS is overriding the linked style sheet.
5. Finally, find a default paragraph in `external.html` and define it with an inline style, such as `<p style="color: #CCC">` and save the template.
6. Reload `external.html` in your browser. Now the paragraph to which you applied the inline style should be light gray, as the inline CSS is overriding the embedded CSS and the linked style sheet. Any other default paragraphs should still be dark gray based on the embedded style. Another method of using the cascade is in the form of multiple external style sheets.

You have already seen how to link to one or more external style sheets for various platforms (such as printers and mobile devices), and this approach is similar, except all the external files here are specifically for the screen:

```
<link rel="stylesheet" media="screen" type="text/css"
href="css/screen/one.css" />
<link rel="stylesheet" media="screen" type="text/css"
href="css/screen/two.css" />
<link rel="stylesheet" media="screen" type="text/css"
href="css/screen/three.css" />
```

Imagine that each of the three style sheets features a rule called `#header`. The declaration for `#header` in each style sheet features the same properties (say height, width, and color), although the value of each is different. In this instance, the browser will consider the last linked style sheet (`three.css`) as most important and perform any rules it contains. Any rules not defined in `three.css` will be performed from the second style sheet (`two.css`). Finally, the browser will perform any remaining styles from `one.css`. Always remember that the later a rule is specified, the more weight it is given.

The hierarchy and cascade is also present within the imported style sheets. As with the previous examples, it's all about the order in which the style sheets are specified. We have seen modular CSS, where the CSS for a site is organized into relevant style sheets. Here's a similar example where four modules imported via a master external style sheet called `external.css`. `external.css` contains the following lines:

```
@import url("default.css");
@import url("layout.css");
@import url("navigation.css");
@import url("forms.css");
```

`forms.css` is highest in the hierarchy, whereas `default.css` is the lowest. Let's assume that in `navigation.css` (second in the hierarchy) there is a class called highlight, used to render text red. Let's also assume that highlight appears in `default.css`, but is used to render text orange. As `navigation.css` has more weight due to its place in the hierarchy, the rendered text will be red. Yet still, `forms.css` isn't necessarily top of the tree. Remember that these style sheets are being imported via a master external style sheet.

Always at the bottom of the cascade hierarchy, the browser's default style sheet will have default settings for headings, paragraphs, lists, and all common HTML elements. It is the browser style sheet that makes links blue and visited links purple unless something different is specified in the sites style sheets. View a page in the browser that isn't styled using CSS, and you will see the default browser styles. As long as you define all the elements you wish to control, their defaults from the browser CSS will be overridden.

2.3 Grouping

Another very important principle for writing well organised CSS is grouping. To illustrate grouping, let's consider the following example,

```
/* Heading styles */
h1 {
font-family:Helvetica,Arial,sans-serif;
line-height:140%;
color:#333;
}
h2 {
font-family:Helvetica,Arial,sans-serif;
line-height:140%;
color:#333;
}
h3 {
font-family:Helvetica,Arial,sans-serif;
line-height:140%;
color:#333;
}
```

Note that aside from the selector, every other rule applied to the selectors are exactly the same. A more efficient way of doing the same thing would be to group the three rules to save space and time.

```
/* Heading styles */
h1, h2, h3 {
font-family:Helvetica,Arial,sans-serif;
line-height:140%;
color:#333;
}
```

What if you wanted to apply one or a few extra rules to one of the grouped selectors but everything else is the same. Again the efficient way would be to group them and define the common rules and then defining the selector again and putting in the extra rules

```
/* Heading styles */
h1, h2, h3 {
font-family:Helvetica,Arial,sans-serif;
line-height:140%;
color:#333;
}
/* Additionally, render all h1 headings in italics */
```

```
h1 {
font-style:italic;
}
```

2.4 Inheritance

Inheritance describes the method by which HTML elements inherit style properties from a parent element. By not declaring a particular CSS value for the child element, that child element may in many circumstances inherit a property given to its parent element. While CSS cascades styles, HTML inherits them. Though inheritance can be a blessing for unaware developers it might turn into a curse. It has a tendency of causing confusion across multiple style sheets, especially when debugging your CSS, and it is strongly recommended that you are aware of it.

To explain inheritance, let us consider a few HTML elements as parents, and the elements within them as their children. The parent passes its properties down to the child, moving through the HTML markup, it is understood that some child elements to a certain parent element can also act as parents to other child elements and so on. This containment hierarchy is referred to as the tree. To illustrate inheritance, let's again go back to headings. The <h1> heading in this example is styled with a very simple rule:

```
/* Top-level heading */
h1 {
color:#333;
}
```

The rule is pretty simple, and you can assume that the heading will be rendered dark gray. Now let's assume that in the HTML the markup dictates that a few of the words should be emphasized using :

```
<h1>This is the best heading <em>in the world</em></h1>
```

At this stage, no CSS rule has been written to manipulate the element. But the text within the element will inherit the colour from the h1 rule (its parent element) and will also be dark grey. To overrule this inherited colour, you can simply define the in the style sheet:

```
/* Make emphasized text shine brightly */
em {
color:#F00;
}
```

Now all the emphasised text across the site is rendered in red, and the element no longer inherits its colour from the parent element. Unless otherwise defined in your em selector, other declarations will still be inherited from the parent.

It is strongly recommended that all major CSS design implementations begin with a <body> element declaration in the main style sheet. The <body> element is the parent of every visible element in your template, and therefore every element without a specified rule inherits from it.

```
<body>
<h1>Absolutely everything else!</h1>
<p>Yep, every visible element is contained within the
body.</p>
<p>And so on.</p>
</body>
```

For example:

```
/* Define all main values for the web site */
body {
margin:10px;
font-family:Helvetica,Arial,sans-serif;
background:#CCC;
color:#000;
}
```

As a result of these declarations, every other rule in the CSS will inherit the values unless specified otherwise. So all headers, paragraphs, lists, and other elements will be rendered with black text (`color: #000`) using the first available font from the suggested options on the end user's machine (`font-family: Helvetica, Arial, sans-serif;`) unless the selector for each child element specifies otherwise, or that child element is housed inside a more immediate parent (such as a column or container) that contradicts the inherited values from body.

Note that some style properties are not inherited from the parent element, the background property being the main example. The child element does not actually inherit the light grey background (`background: #CCC;`), but rather the parent element's background appears by default. In other words, the child element can be thought of as

having the inherited font colour, but it should not be assumed that it has the same background colour.

2.5 Contextual selectors

In the previous section, we used emphasised text in a heading and illustrated basic inheritance. The em selector was added to ensure the element in the HTML would be rendered with red text. Let's use the same CSS code again:

```
/* Top-level heading */
h1 {
color:#333;
}
/* Make emphasized text shine brightly */
em {
color:#F00;
}
```

The downside of this is that all emphasized text across the whole web site would also become red, regardless of its parent element. Assuming the rule is only meant to target the element when a child of <h1> headings, a simple adjustment can be made to put the emphasized text into context:

```
  /* Make emphasised text shine brightly ONLY in an h1
heading */
  h1 em {
  color:#F00;
  }
```

Contextual selectors are basically two or more simple selectors separated by whitespace. We construct a contextual selector to show that the rule will only have an effect when the last selector (em) is a direct descendent (be it a child, grandchild, great grandchild, or so on) of the first selector (h1).

```
<h1>This is the best heading <em>in the world</em></h1>
<p>I'm sorry but it simply is <em>not</em>, you fool.</p>
```

The element owned by the <h1> element will be red, whereas the one owned by the paragraph will not.

2.6 CSS measurements

One of the properties of a CSS-based design is its ability to be very flexible. As a designer, you would want your carefully designed layouts to look exactly the way you imagined it across all platforms and for every end user.

This can lead you to a very tempting notion of defining exact dimensions for everything. But this is not always the best way to go about things. You should be aware of the inherent flexibility of the web as a viewing platform, and that there are a multitude of ways a web page is viewed – different screens, different browsers, different devices and different screen resolutions.

Two options for measurement that are available through CSS are absolute and relative. As long as web designers apply a measure of control in their style sheets, relative measurements actually do a much better job of tightening the design while also making it full proof without pitfalls in terms of display.

2.6.1 Absolute measurements

Absolute values are fixed, specific measurements. They let you be exact in your measurements and control the display of your web pages. Absolute values in CSS are inches (in), centimeters (cm), millimeters (mm), points (pt), and picas (pc). Absolute measurements will only give the results based on the fact that the browser interprets the number of pixels on the screen and the dimensions of the screen. Often, these might be wrong. Not all computers know how big the screen is. In these cases, the browser makes a guess and assumes a size. But it is good to note that almost all generic day to day CSS design doesn't require absolute measurements, so let's skip to the implementation of relative measurements.

Note - A point (pt) is 1/72 of an inch and a pica (pc) is equivalent to 12 points, or a sixth of an inch.

2.6.2 Relative measurements

Relative measurements unlike absolute values are calculated in comparison to an inherited value. The relative values in CSS are percentage (%), x-height (ex), ems (em), and pixels (px). Our web sites will primarily be viewed on computer screens, but you can never be sure about the screen resolution or the browser. Some end users might view your site on PDAs, cellphones, projectors, or even on HDTVs. Consider also that many users manually configure their devices to suit their own preferences (like screen resolutions), so it is necessary that your design is not compromised in these situations. A very important thing to maintain in web design is accessibility, ensuring that all your visitors can access all the content you wish to provide.

Relative measurements are proven to provide the best solution for text resizing. While relative values give the designer less control, they do create a better experience for the end user.

2.6.2.1 Pixels

Pixel measurements give designers the most control over their layouts. It is the closest relative measurement to absolute measurements. Hence it is the most consistent unit of measurement; pixels are most commonly used for declaring the margin, padding, borders, height, and width of elements and can be successfully combined with other relative measurements for all kinds of layout requirements.

Pixels are not, however, the best option for sizing text, text sized with pixels will ignore user preferences, and so if a user has set his or her browser to always show text at 20 pixels, this will often be overridden by your pixel-based declarations.

2.6.2.2 Percentage

The percentage measurement provides a lot of flexibility. Percentage values are always relative to another value, such as a width or height declared in the parent element. In other words, a percentage can only be declared in relation to a previously defined size, within the parent element, or is based on the width of the browser window. Percentage is an integral ingredient of liquid designs where pages and their elements stretch to fit the browser window.

But some caution is necessary when it comes to percentage values, where inheritance causes problems. While pixels and ems retain some amount of control, the results of percentage values can end up very different to what is expected. When calculated by the browser, the layouts and elements can be misplaced or shunted underneath each other.

In the following example, the percentage value is given for line-height, used to control the distance between the lines of text, and relates directly to the font-size value within the rule. In this case, line-height will be applied to the text as a pixel value that is 120 per cent of 10px, which gives a line height of 12px.

```
/* Define default paragraph values */
p {
font-size:10px;
line-height:120%;
}
```

2.6.2.3 Ems

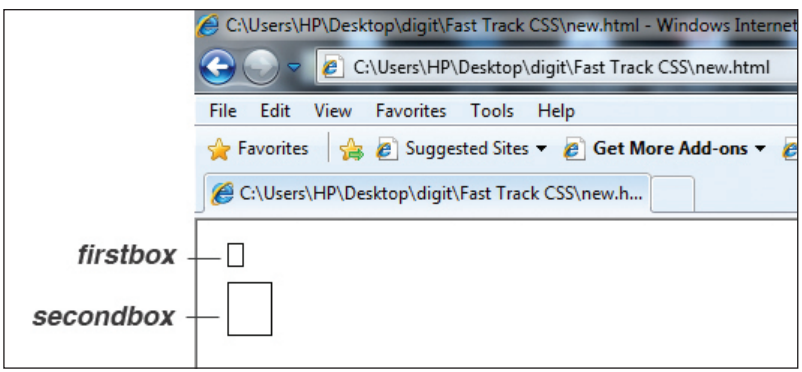
The em is the most misunderstood unit of relative measurement in CSS. But it is, by far, the most flexible and ideal mode of measurement in designing,

when the end user's viewing device and text preferences are unpredictable. The origin of the word itself is unconventional. It stems from the typography industry roughly equaling the size of an uppercase letter M, hence it is pronounced "emm." However, in reality an em is actually larger than that. Because some fonts don't even have an M in them, the term has come to mean the height of the given font.

The em CSS measurement can be employed to define the lengths of almost any CSS property. This makes it especially powerful when applied to fonts and their containing elements. The user can resize the text using his browser tools, and the containing element defined in ems scales with it.

To illustrate this say within a given element, the font-size is set to 11 pixels. Then one em is equal to 11 pixels. If the font-size of another element is 30 pixels, one em in that element is equal to 30 pixels. If two rules for basic containers are identical except for the font-size, which is 11px and 30px, respectively. Both rules are given a width and height of 1em.

```
/* First container for the em example */
#firstbox {
margin:10px;
font-size:11px;
width:1em;
height:1em;
border:1px solid black;
}
```



The two containers will be of different sizes when viewed in the browser

```
/* Second container for the em example */
#secondbox {
margin:10px;
font-size:30px;
width:1em;
height:1em;
border:1px solid black;
}
```

Although both boxes have a height and width of 1em, the two different font sizes make one box larger than the other. The first box is 11px × 11px, and 1em × 1em. The second box is 30px × 30px but is also 1em × 1em. Thus each resulting em measurement is relative to whatever contains it. Further rules within #firstbox would be free to define measurements based on the em, knowing that it is equal to 11px.

3 Building blocks

Margin, padding, borders, widths, heights etc. are all familiar terms and they do what you would expect them to do. One important aspect of CSS design is the use of divisions in providing greater flexibility and marking out regions within the page.

3.1 Divisions (Divs)

 elements, widely referred to as divs are HTML elements used to define areas within an HTML page. A division can encompass everything else within the <body> of the page or more typically be used for distinct areas such as headers, footers, navigation bars, and so on. Often, the HTML in your templates does not provide enough hooks to attach your styles. For example you wouldn't be able to turn paragraphs or headers into boxes directly. This is where divisions (henceforth referred to as divs) come into picture.

A div is created within HTML by wrapping content with the tag. The result is a hook to which CSS can be applied.

```
<div id="container">
<p>This is our content area.</p>
</div>
```

Let's apply some simple CSS to the container ID:

```
/* Container holds all visible page elements */
#container {
padding: 20px;
border: 1px solid #000;
background: #CCC;
}
```

We have already seen how IDs and classes are used as identifiers to a standard HTML element. The same formula is used for divs by referencing the selector in the opening tag using id="name" or class="name". In this case, we've used an ID named container to define the division:

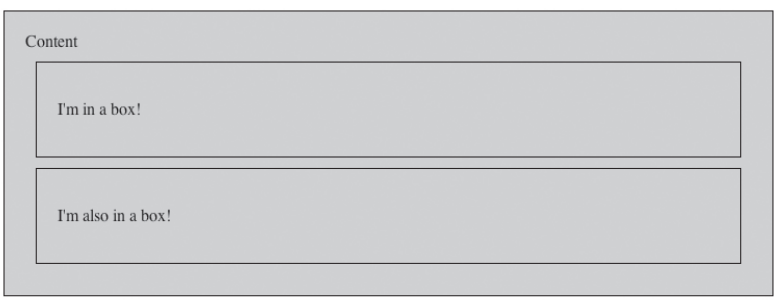
The container div in the browser

```
<div id="container">
<p>This is our content area.</p>
<div class="box">
<p>I'm in a box!</p>
</div>
<div class="box">
<p>I'm also in a box!</p>
</div>
</div>
```

The CSS rules for the box class are almost the same as that for the parent container, except for the background colour, which will appear by default. Note that, as no set width is defined for the box, it will stretch to fit whatever contains it, be that another div or the browser window:

```
/* Define styling of our reusable box */
.box {
margin: 10px;
padding: 20px;
border: 1px solid #000;
}
```

Our three divisions are now clear in the design. Remember that each div can contain any number of elements, be they headings, paragraphs, images or more divs.



The container div now has two boxes

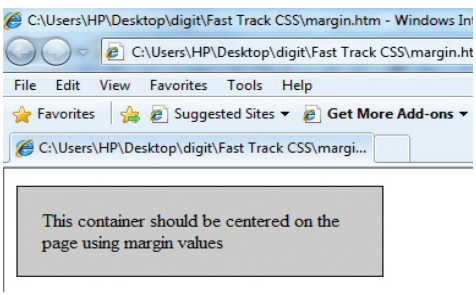
3.2 Dimensions: width and height

Two very important properties that can be declared for any element are its width and height. Both properties are essential for setting specific heights and widths of elements. Consider that an element's width will increase to fit its container, while its height will increase to encompass its content, you will start to realise before long that, in some situations, a certain amount of control is missing. By applying width and/or height rules, you can regain this control.

Values can be given as a length, percentage, or auto. Note that all of these values can be influenced adversely by other rules within the style sheet, and also by the HTML elements they might contain. For example, the resulting display can be affected by a number of values from margin, padding, border, or child elements.

3.3 Margin

The margin property is used to declare the margin between an HTML element and its neighbours. The margin can be set for the top, left, right, and bottom of the given element. Note that margin values are not inherited from parent elements. There are three choices of values for the margin property, which are length, percentage, or auto. Note that if the value is `0`, you do not need to add `px`. Consider the following CSS for a container div. Note that it has a fixed width of 300px and no margin properties.



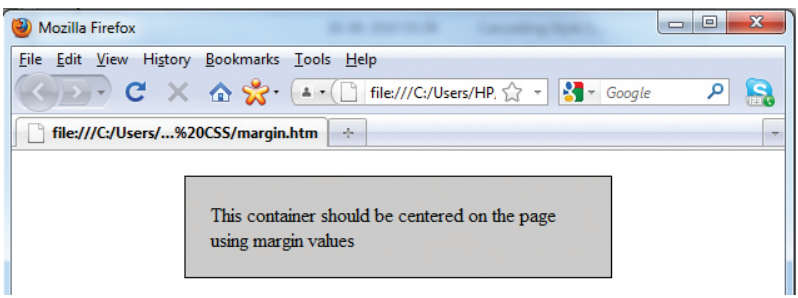
The container is always starts from the left of the page by default

```
/* Basic container */
#container {
width:300px;
border: 1px solid #000;
padding: 20px;
background: #CCC;
}
```

By applying margin properties to each side of the container, the display can be significantly altered:

```
/* Basic container */
#container {
width:300px;
margin-top: 20px;
margin-left: auto;
margin-right: auto;
margin-bottom: 1em;
border: 1px solid #000;
padding: 20px;
background: #CCC;
}
```

The container is now 20 pixels from the top edge of the <body> element, and is centred due to its set width combined with left and right margins



The margin:auto values centre the container with respect to the browser window.

set to auto. A couple of easy shortcuts are available to reduce up to four margin:value declarations into one.

```
/* Basic container */
#container {
margin: 20px auto 1em auto;
}
```

The order of the values is very important here. The order is top (20px), right (auto), bottom (1em), and left (auto). The best way to center an element using CSS is to use the auto value for left and right margins. For modern browsers, all this requires is a set width rule and the left and right margins given the auto value.

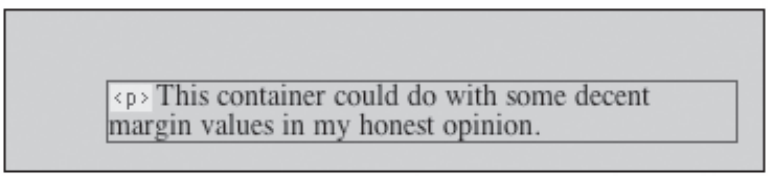
3.4 Padding

Padding is the distance between the edges (borders) of an HTML element and the content within it, and can be applied to any element. Both length and percentage values are available. But there is no auto value, and negative values cannot be declared for padding. Let's take the container div again and this time, add custom padding to each side:

```
/* Basic container */
#container {
width:300px;
margin-top: 20px;
margin-left: auto;
margin-right: auto;
margin-bottom: 1em;
border: 1px solid #000;
padding-top: 20px;
padding-left: 10%;
padding-right: 1em;
padding-bottom: 0;
background: #CCC;
}
```

Percentage in this case refers directly to the parent element's width. So if padding-left: 10% is declared, that equates to 10% of the parent element's given width. Using ems would also allow the padding to scale proportionately with the element. Such values come in very handy for liquid layouts.

The same shortcuts used for margin values are also available for padding.



The padding applied to the container positions the paragraph accordingly

As with the margin property, order is top (20px), right (1em), bottom (0), and right (10px).

```
/* Basic container */
#container {
padding: 20px 1em 0 10%;
}
```

Likewise, if all four values are the same, the padding declaration can be further shortened:

```
/* Basic container */
#container {
padding: 20px;
}
```

3.5 Border

Borders are a simple concept, but have numerous possibilities. Any element can have a border placed around it, and borders can be placed on all sides, or just the sides you desire. The border property is particularly flexible as each border can be of a specific width, colour, or style. Numerous values can be applied and a number of shorthand declarations are also available.

The default values are a border with a medium thickness and inheriting the text colour of the parent element. Only by applying further values can this default state be influenced. The full list of border properties includes

```
border-style
border-width
border-top-width
border-right-width
border-bottom-width
border-left-width
border-color
border
border-top
border-right
border-bottom
border-left
```

Controlling borders is relatively easy, but it is worth looking at each property in a little more detail.

3.5.1 border-style

The browser must first understand the style of border to be drawn before moving on to further border declarations. In other words, the style keyword is declared before any other property of the border. The property applies a defined style to one, several, or all borders. Values for border-style are none, dotted, dashed, solid, double, groove, ridge, inset and outset.

3.5.2 border-width

More specifically, in this section we will look at border-width, border-top-width, border-right-width, border-bottom-width, and border-left-width. These properties allow you to define the width of the element's edges one by one or all at once. Note that for a border-width value to be applied, a border-style must first be declared. Several keyword values are available here, as well as relative lengths.

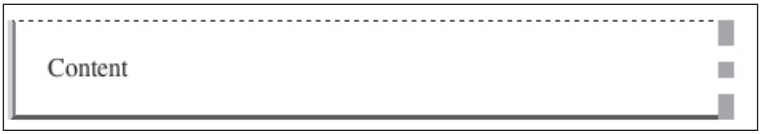
The values for border-width include thin, medium, and thick, although caution is required as different browsers render these borders in different ways. Exactly how many pixels each browser will use varies. Note that if you define a border-style but not a border-width, the default value is medium.

3.5.3 border-color

Remember that, unless you declare the colour for the border, it will inherit colour from the element or parent element. As there is only one property for colour (border-color), multiple colours for different sides must be declared using shorthand:

```
/* Container for cantering all our content */
#container {
width: 400px;
margin: 10px auto 10px auto;
padding: 20px;
border-style: dashed dotted solid ridge;
border-top-width: thin;
border-right-width: 20px;
border-bottom-width: medium;
border-left-width: thick;
border-color: #000 #999 #333 #CCC;
}
```

This gives us borders from black (top) to lightest grey (left)



Container with different borders

Applying simple borders to your divisions and other key elements is a brilliant way of creating a wire frame. Wire framing a design with a thin solid or dashed line around your divs can help you understand how one element relates to another, and also identify problems with alignment and juxtaposition. You can apply a simple dashed line to all divs, for example, using a base selector for the tag:

```
/* Place a thin gray border around all divisions */
div {
border: 1px dashed #CCC;
}
```

This rule would be placed immediately after the body selector in the style sheet, and would ensure all divs you create inherit a thin grey border, unless you specify otherwise in that element's rule. To apply this rule to further base elements, simply group the selectors together:

```
/* Place a thin grey border around the following elements
*/
div, h1, h2, h3, h4, ul {
border: 1px dashed #CCC;
}
```

d

4 Text properties

Arguably the most important element of any page is the text on it. The text represents the content you are portraying through your website. Often, all the effort in styling goes into some incredibly beautiful masthead, logo, or background, and making the top of the web page look great, but scroll down to the actual content you might encounter a CSS famine.

The first thing that appears when a page is loading is the text on it, and the presentation of text has the power to convey to the user if the website is serious or friendly, modern or traditional, formal or casual. Choosing the right font or a combination of fonts is important in creating the right first impression.

The first thing you should generally do is override some of the browser's default CSS properties, specifically the typeface and the font size. This is fairly straightforward, but there are a few issues you should be aware of. The following properties are typically declared in the body selector, allowing all the following elements to inherit the values. Further customization can be done to individual elements if required.

4.1 font-family

The font-family property declares a list of font family names and/or generic family names for the element, in a priority order. The browser will use the first available font on the end user's machine. There are two types of font-family values:

Family name: The name of a font family, like Times, Georgia, Arial, etc.

Generic family: The name of a generic family, like serif, sans-serif, cursive, fantasy, etc.

The values should be separated by a comma, and it is suggested that you use a generic family name as the final option in the list. If a family name contains whitespace it should be enclosed in quotes. For example the font Lucida Grande is written as "Lucida Grande". Note that single quotes need to be used if you are embedding the style in HTML.

In the example that follows we define the font-family for the body selector; this will ensure all the elements in the page will inherit the same font family unless declared specifically.

```
/* Specify blanket rules for all elements */
body {
font-family: "Lucida Grande", Arial, Sans-serif;
}
```

Lucida Grande is specified first, and since it has a whitespace character it is contained within quotes. Any machines with Lucida Grande will display text with this font, and if it isn't available, the display will default to Arial. If neither Lucida Grande nor Arial are available, the browser knows to use whatever appropriate sans serif font it can find next, as the generic family sans-serif is specified.

4.2 font-size

We have already seen various uses of the preferred measurements, namely pixels, ems, and percentage.

```
/* Specify blanket rules for all elements */
body {
font-family: "Lucida Grande", Arial, Sans-serif;
font-size: 12px;
}
```

By declaring font-size:12px in combination with the font-family declaration, you will ensure that all elements will be sized to 12px regardless of any inheritance (unlike ems, which can be influenced heavily through cascading rules). However, there are a few exceptions to this blanket font-size declaration. Note that all headings (<h1>, <h2>, <h3>, and so on) will retain the default font sizes declared by the browser style sheet unless you redefine them specifically.

Again there are a few useful methods of shortening font declarations, pulling several into one simple statement. Later you'll combine four or five declarations into one, but for now, let's collate font-family and font-size. The most important thing with shorthand is the order in which declarations are stated. In this case, font-size precedes font-family.

```
/* Specify blanket rules for all elements */
body {
font: 12px "Lucida Grande", Arial, Sans-serif;
```

```
}
```

4.3 Available fonts

The most common problem a designer faces with regard to the Web is the poor choice of fonts available on the user's computer. This boils down to the fact that the only fonts that can be safely used are the ones commonly installed on every computer used to view the web site. For example, just because you have Sharktooth Italicized installed on your computer doesn't mean everyone else has it as well. If you use such a rare font in your style sheet, and view the text on your web site using your computer, it will be rendered with it, but very few other users will have such an obscure font installed and available to their browser. Thus, it is important to think about web-safe fonts.

A few web safe fonts

4.4 line-height and letter-spacing (Kerning)

It is good setting the line-height in the body selector, as all elements can benefit from inheriting this value. Line height is the space between two consecutive lines. Let's say that the spacing between the lines of text will be the given a certain percentage of the current font-size. So, a line-height of 100% will make no difference, whereas a line-height of 150% will create a space half the size of the font. A line-height of 200% will create a space equal to the size of the font, and so on.

The spacing of characters is called kerning. CSS allows you to emulate this tight text control with the letter-spacing property. Where line-height creates extra whitespace between lines of text, letter-spacing is used to adjust the spacing between characters. Again, 'normal' can be declared to override inherited letter-spacing, but mostly you will declare letter-spacing in pixels. In the following example, letter-spacing is declared for the grouped headings:

```
/* Specify blanket rules for all elements */
body {
margin: 10px;
```

```
border: 1px solid #000;
padding:10px
font: 12px Verdana, Arial, Sans-serif;
line-height:150%;
}

h1, h2 {
letter-spacing:3px;
}
```

It is worth noting that unlike line-height, negative values are allowed with letter-spacing, so something like 'letter-spacing: -0.5em' can be used to bunch up the characters if required. Using the em measurement will ensure that the spacing scales if the text size is later increased.

4.5 font-weight and font-style

The font-weight property is used to set the thickness of your text. Typically, the declaration is either normal or bold, but some browsers do support numeric values in increments of 100. That are 100 (lightest), 200, 300, 400 (same as normal), 700 (same as bold), 800, and 900 (even bolder!).

The default for font-style is normal, but typically you would use this property to declare any text that needs to be rendered in italics. Values are normal, italic, and oblique.

4.6 font-variant

The font-variant property is used to display the text in a small-caps font. Lowercase letters can be converted to uppercase letters, but all the letters in the small-caps have a smaller font size compared to the rest of the text. This is used to depict less important info such as stats, figures, or footer information. Possible values are simply normal or small-caps. Note that the browser will use a proper small-caps font if available; otherwise the effect is done computationally.

4.7 text-transform

This property is sort of an opposite of font-variant, where all characters can be rendered uppercase without reducing the font size. The 'text-transform:uppercase' declaration is especially useful for headings, where it is semantically incorrect to type using uppercase characters in the markup.

The other key text-transform value is 'capitalize', which ensures that the first character of any word is rendered as an uppercase character, which again is very useful for headings or lists. Other text-transform values are 'none' and 'lowercase', the latter removes all instances of uppercase characters.

Taking this core template, and mixing it up the CSS, you'll see a myriad of declarations across the selectors. Note that shared heading declarations are grouped, whereas individual ones are defined individually:

```
/* Specify blanket rules for all elements */
body {
margin: 10px;
border: 1px solid #000;
padding:10px;
font: normal 12px Verdana, Arial, Sans-serif;
line-height:150%;
}

p {
font-variant:small-caps
}

h1, h2 {
letter-spacing:1px;
}

h1 {
font-family: Georgia, Times, serif;
text-transform:uppercase;
}

h2 {
font-family: "Helvetica Neue", Arial, sans-serif;
text-transform:none;
font-style:italic;
}
```

This was a good example as a means of showing how to apply the various font values in one place, but in the real world it is important to make font decisions for a reason. Italicized text is considered less legible by many, so ensure you are using it because you have to.

4.8 Font shorthand

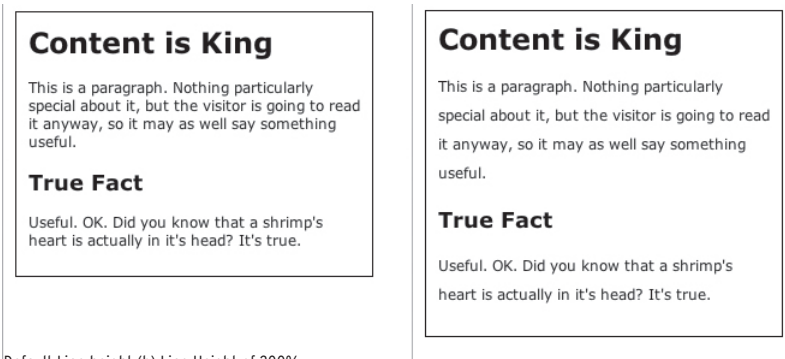
The previous example has a CSS of up to 23 lines. This can be greatly reduced by making use of font shorthand. While letter-spacing and text-transform cannot be included in the shorthand, all other font properties including line-height can. In the following example let's present all the font properties in one declaration. As long as the order of values is correct, the display will be exactly the same as in the previous example. First, consider the following selector for a paragraph:

```
p {
font-style:italic;
font-variant:small-caps;
font-weight:bold;
font-size:12px;
font-family:verdana,arial,sans-serif;
line-height:150%;
}
```

Building upon the basic shorthand we have seen with font-size and font-family, values can be combined to bring everything into one declaration. Note that in the illustration below after font: the value order is identified by the name of the property. The order is very important:

```
p {
font: style variant weight size/line-height family
}
```

Now, replace each word with an actual value from the original six lines of CSS, Values can be omitted if need be, so if you wish not to declare the font-variant, simply leave it out. As long as the order of declared properties



Default Line height (b) Line Height of 200%

doesn't change, it will work.

```
p {
font: italic small-caps bold 12px/150% verdana,arial,sans-
serif
}
```

4.9 Indenting Paragraphs

Indenting the first line of a paragraph is something you are very familiar with. And unwitting many tend to do it by adding whitespace before the text. Indenting each paragraph is especially useful where line-height is quite narrow. It helps in drawing attention to the start of the paragraph. CSS gives us the text-indent property to do exactly that.

```
/* paragraph styling */
p {
font: 12px verdana,arial,sans-serif;
text-indent:15px;
}
```

The result of this is a 15px text-indent for the first line of the paragraph.

5 Colours, backgrounds and images

The very essence of CSS is that it allows us to separate the presentation from content. The entire core content including the text and key images are locked in the HTML, leaving all the actual styling separate in a style sheet. Therefore, the style sheet is where all the design material is specified. This includes the colour and placement of decorative images. CSS allows you to apply colour to text and containers, the application of background images etc.

Background images are key to realizing the creative potential of CSS-based web design, and they literally transform your pages into works of art if used properly. It should be noted that adding any images to a page, whether inline or with CSS, will increase its weight, and therefore the time it takes to download. Many designers do more harm than good using CSS to apply unnecessarily large photographic backgrounds and complicated images. This when combined with other heavy ingredients such as JavaScript, significantly increases a page's total size. The trick is to use background images sparingly and creatively. It also helps to be aware of occasions where the same image can be reused and repositioned on a page.

Advanced CSS deals specifically with incredibly ambitious background image use, but these processes can be slightly complex. In this chapter, you will first learn the correct ways of applying colour to your web pages and later begin to work with background images in a sensible way.

5.1 Colours

Let us work with the notion that there are only 216 colours that you may use on your web pages. Early web designers were bound by many rules that restricted their choices. Among the many limitations, one of the most widely adopted was the concept of a web-safe colour. 216 colours were guaranteed to be displayed correctly on any platform the page is displayed in, and the problem of using any others was that they could rendered differently on different browsers or not rendered at all. But fortunately most modern browsers allow for millions of colours.

5.1.1 Web-safe Colours

Web-safe palettes or browser-safe palettes consist of 216 colours that display solidly, consistently, and without graduation on any monitor capable of

displaying at least 8-bit colour (256 colours). The reason why this palette doesn't contain the maximum of 256 colours is that only 216 of these are exactly the same on all computers the remaining 40 vary in the traditional Mac and PC. So it is recommended that we stick with these 216 colours keeping old browsers and computers in mind. It is not necessary to do so, but

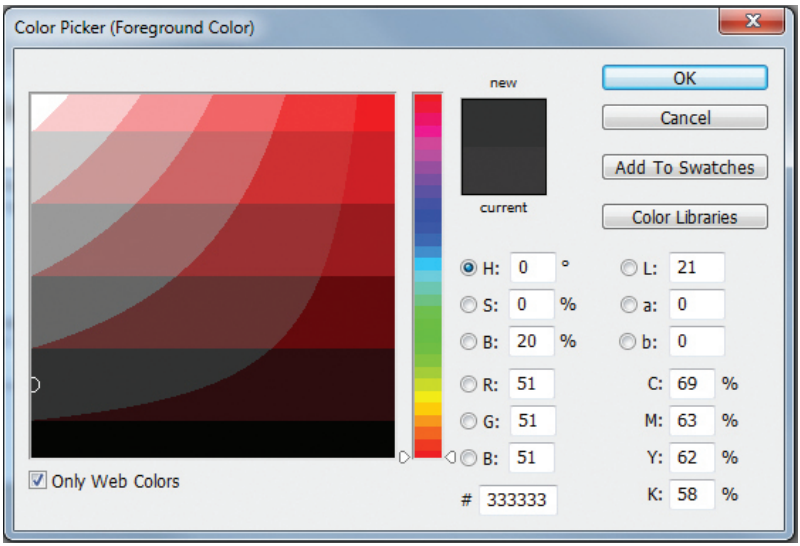


Figure 6-1: Web Safe Colours

why use other colours when you can get your work done with the traditional web safe colours.

5.1.2 Specifying Colour

Colours can be specified in a number of ways. Many specify colour as an RGB triplet in a hexadecimal format (a hex triplet). Others often use their common English names in most cases. It is also possible to use RGB percentages or decimals. The following examples are all valid for the colour red:

```
#f00
/* #rgb */
#ff0000
/* #rrggbb */
```

```
red
/* common English name */
rgb(255,0,0)
/* integer range 0 - 255 */

rgb(100%, 0%, 0%)
/* float range 0.0% - 100.0% */
```

The CSS language defines the same 16 named colours as the HTML 4 specifications, plus CSS 2.1 adds the "orange" colour name to the list. The 17 named colours are listed below along with their hexadecimal code for reference

```
black          #000000
navy           #000080
green          #008000
teal           #008080
silver         #c0c0c0
blue           #0000ff
lime           #00ff00
aqua           #00ffff
maroon         #800000
purple         #800080
olive          #808000
gray           #808080
red            #ff0000
```

The developing CSS3 specification also introduces Hue, Saturation, Lightness (HSL) colour space values to style sheets.

The following two examples do the exact same thing,

```
p {
color: #F00
}
p {
color: red
}
```

Colour is an inherited value. Therefore, specifying the colour blue on a containing div will result in all contained text (including headings, lists, block quotes, and so on) being blue, unless the specified element is given a different value.

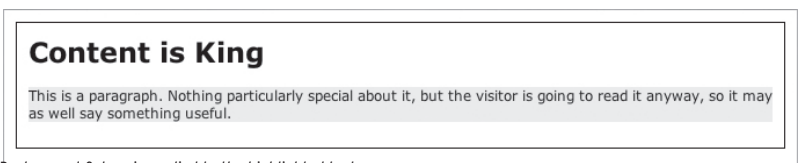
5.1.3 Background colour

The background-color property is used to set the background colour of an HTML element. It is the fastest way of transforming your site from plain black text on a white background into something more engaging. This property is something you will use very frequently. Again, it is recommended that if you decide to give your web site a solid-coloured background, use a web-safe colour. This will guarantee that the colour will not turn out wrong when it displays on other computer platforms.

Just like a highlighter pen, CSS uses background-color property to colour the background of the text. In the following example, a yellow background-color is declared alongside black text:

```
p {
color:#000;
background-color:#FF0;
line-height:150%
}
```

A custom element can be used to control a section of the paragraph, rather than the whole chunk. This method demands additional markup, but it is the only way to affect a designated portion of text contained within a paragraph,



Background Colour is applied to the highlighted text

unless you target text already defined with phrase elements such as or . Rather than assign the background colour to the whole paragraph, a class is created that will be used to cloak the text that needs a highlight. In the following CSS, background-color is removed from the paragraph selector, and a new class called highlight is defined:

```
/* Define basic paragraph style */
p {
color:#000;
line-height:150%
}

/* Highlight class to pick out key words or phrases */
.highlight {
background-color:#FF0;
}
```

5.2 Image formats for backgrounds

Three main formats are acceptable, namely GIF, JPEG, and PNG files. The latter is used considerably less than the other two, but it is still a very useful type when needed.

5.2.1 GIF

The GIF (pronounced "gif" or "jif" by some) format keeps the size of the file as small as possible. The GIF was created when colour displays were limited to 256 colours and the internet was slow. Instead of describing one pixel at a time in terms of its colour, a gif describes the boundaries of an area and the single colour within that area following the concept of vector graphics. In cases where there are large areas of certain colours, the file size is smaller.

The best thing about GIFs is that they can carry a certain amount of transparency, which can be invaluable for web design. Imagine that you wish to have a small arrow icon appear in every heading, but that headings appear on many different background colours. This is where the transparent GIF will come in handy. As long as the GIF is saved with transparency, it will allow the background colour to come through. Regrettably, transparent GIFs are not perfect, and rounded or jagged edges will be saved with a few non transparent pixels around them. While transparent GIFs really extend the limits of image use on the Web, they do have limitations, especially where transparent gradients are concerned. A more complex solution to a transparent image is the "PNG".

5.2.2 JPEG

The familiar image format JPEGs are incredibly flexible, but an image saved as a JPEG (pronounced "jay-peg") isn't analysed for colour in the

same way as a GIF. This is the most commonly used standard method for photographic images.

5.2.3 PNG

The PNG (pronounced "ping") was developed to improve upon the limitations of the GIF, where machines capable of displaying millions of colours were outgrowing a format limited to just 256 colours. This is the chief reason why the PNG is growing in popularity among web designers who desire greater flexibility but maximum image quality retention. Most will resort to the PNG when a much more complex transparent image is required, perhaps one containing a very subtle gradient or shadow. The big problem is that there has been the lack of support from Internet Explorer up to version 6. IE will flatten your transparent PNG and render it as a block, revealing none of the colour underneath.

5.3 Prepare your template and style sheet

To illustrate the flexibility of background images, it is worth preparing a new template to experiment through the rest of this chapter. Create a new template called images.html, and add the following markup:

```
<html>
<head>
<title>Chapter 6</title>
<link rel='stylesheet' media="screen" type='text/css'
href='images.css' />
</head>
<body
<div id="container">
</div>
</body>
</html>
```

Note that the body element contains just one child element, the container div we used earlier in the chapter. That div will act as the sandbox in which just one background image will be obeying your every command as it is shunted around, duplicated, and positioned. Note that a style sheet called images.css is linked from the head of the new template. Create that style sheet and paste the following into it:

```
/* Specify blanket rules for all elements */
```

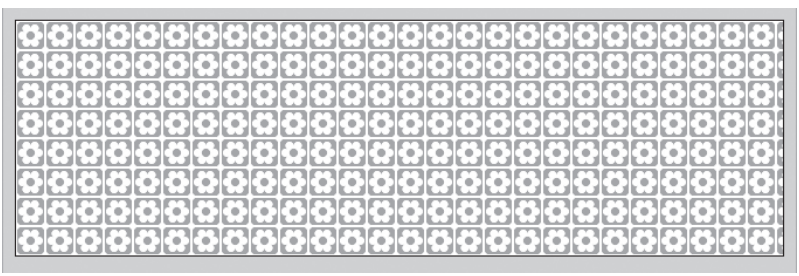
```
body {
margin: 10px;
background-color:#CCC;
}
/* Container for all page content */
#container {
height:200px;
border:1px solid #000;
background-color:#FFF;
}
```

Save the new style sheet. All the properties used should now be familiar to you. Notice that again the body has a gray background and a 10-pixel margin from the edges of the browser window as with the previous exercise. Again, the container is white, but this time has no padding, and has a set height of 200 pixels so that it doesn't collapse due to being empty.

5.4 Background images

The basic property and declaration is very similar to that of background-color. Load `images.html` in your browser. Quickly get a background image in there. Tile images can be repeated infinitely either horizontally, vertically, or sometimes both directions. First, the tile can be made to appear in the container by adding the background-image property to the container selector.

```
/* Container for all page content */
#container {
height:200px;
```



The small image tiles throughout the background

```
border:1px solid #000;
background-color:#FFF;
background-image:url(tile.gif);
}
```

However, the result is that the image doesn't display once, but several times. By default, the background-image property will replicate the specified image as many times as necessary to fill the container.

5.4.1 Repeat

So how would you control repeating of the image? CSS in the form of the background-repeat property allows you to do just that. There are four possible values, 'repeat', 'repeat-x', 'repeat-y', and 'no-repeat', all of which you will inevitably use. It is worth going through this in detail and skipping the repeat value, as that is already happening by default.

To turn off background image tiling, you will need the no-repeat value. Adjust the container properties as follows:

```
/* Container for all page content */
```

Only one instance of the image is rendered

```
#container {
height:200px;
border:1px solid #000;
background-color:#FFF;
background-image:url(tile.gif);
background-repeat:no-repeat;
}
```

This simple declaration will ensure that the image only displays once, and that it will be placed immediately inside the container, at top left. The tile can be allowed repeat either horizontally or vertically only. This is an essential technique allows for great flexibility. In the CSS, change the background-repeat value.

The small image repeats along the Y axis

```
/* Container for all page content */
#container {
height:200px;
border:1px solid #000;
background-color:#FFF;
background-image:url(tile.gif);
background-repeat:repeat-y;
}
```

The tile will only be displayed vertically for the full height of the container. This method is ideal for creating borders for containing elements, or for using a wide repeating strip as the main page background.

5.4.2 Position

By default the background-image property places the image (or starts the tiling process) at the top left of the container. More flexibility is available to the background-position property, which allows you to specify exactly where the image should be placed in relation to its container. If you wish for the image to appear just once, and at the right of the container, simply specify 'background-repeat:no-repeat' followed by 'background-position:top right'

The different values possible are,

```
top left
top center
top right
center left
center center
center right
bottom left
bottom center
```

Using background position you can position the image anywhere in the background

```
bottom right
x-% y-%
x-pos y-pos
```

If the general English values are too vague for what you exactly want, you can specify the exact coordinates in pixels or as percentages instead or even combine them with the English values. Look at the following examples:

```
background-position: 50px left;
background-position: 10% 50%;
background-position: 10px 20px;
background-position: 20px bottom;
```

5.4.3 Attachment

The background-attachment property is not very commonly used, but is still very effective. A fixed background image declared in the body selector ensures that the background image stays stationary while the main container and everything inside it scrolls as be expected. Two values are available, scroll (the default) and fixed. This method does require a certain amount of caution and careful preparation of graphics for some situations, but the results can be very spectacular.

5.4.4 Background Shorthand

With so many background properties available, shorthand is very useful to combine several background values into one line of CSS. In the previous example, rules for background-color and background-image have been specified. These two rules can be combined by specifying the background property, and then the values in their correct order (colour then image).

```
background:#FFF url(tile.gif);
```

If the image needs to be displayed to the right, and at the top of the container, the background-position value can be placed at the end of the list of values.

```
background:#FFF url(tile.gif) right top;
```

As long as the order is correct, the shorthand will work. You now have three lines of CSS combined into just one. 'Color, 'Image', 'Position', and 'Repeat'

If you wish for the image to display just once, calling for no-repeat. The repeat value comes next in the list.

```
background:#FFF url(tile.gif) right top no-repeat;
```

Remember that any of the preceding values can be removed, as long as the order remains the same. If you remove the background-image value, the other image values will be worthless.

6 Lists

The fact is that the list is a very simple yet essential tool for organizing data, it is incredibly useful for the web designer. View the source code of any web site built using web standards, and you are almost guaranteed to see a list for the navigation buttons, a list for the external links, and probably a list for any buttons or arrays of data. Utilizing a list at an early stage of a web site project ensures the design will remain flexible and functional in any viewing scenario.

In this chapter, you'll be quickly reminded of the basic list markup and assess the different kind of lists that HTML provides, particularly unordered and ordered lists. We will learn to apply ID and classes to lists to attain greater control and then take things a bit further with nested lists controlled first with IDs and classes, and then manipulated with no extra markup, taking their basic hierarchies as hooks for more complex CSS control.

A list of items can be created in numerous ways in HTML, although all may not be semantically correct. The bad methods include adding a `<br />` tag after each item, or treating each item as a paragraph. The correct approach is to use an ordered or unordered list element, using the simple `<li>` tags for each item. The major benefit of this approach is that your list will be displayed as a numbered or bulleted list without the use of CSS, and is considerably easier to control with CSS. Using semantically correct list markup also makes it easy to single out items within a list, and also makes nested lists easier to manage.

6.1 Unordered list

Let us create a new template called `lists.html` and add the following markup inside the empty body element. Notice that the unordered list is placed inside a container, which will act as a hook for more CSS later in this section.

```
<div id="container">
<ul>
<li>Drinks Menu</li>
<li>Beer</li>
<li>Spirits</li>
<li>Cola</li>
<li>Lemonade</li>
<li>Tea</li>
```

The disc bullets appear by default

```
<li>Coffee</li>
</ul>
</div>
```

This list features an initial list item "Drinks menu". Semantically speaking, this is the heading for the list (such as `<h3>Drinks menu</h3>`), but for the purposes of this chapter, it is declared as a list item so that you can see how it can be treated differently. This approach will make more sense when we look at nested lists later in the chapter.

Before moving on, let's first work through some of the basic CSS list properties. By default, the list is bulleted with small discs, emulating a typical list such as you might find in Microsoft Word. Note also that even though the container has no internal padding, the list is still placed well away from the left edge—approximately 30 pixels away. This padding is actually the distance between the left edge of the unordered list and each item it contains.

6.1.1 list-style-type

The list-style-type property allows you to specify one of a number of possible markers instead of the default disc for each list item. There are numerous values available, many of which you won't need (Hebrew or Armenian characters, for example), but the following five values may well be useful:

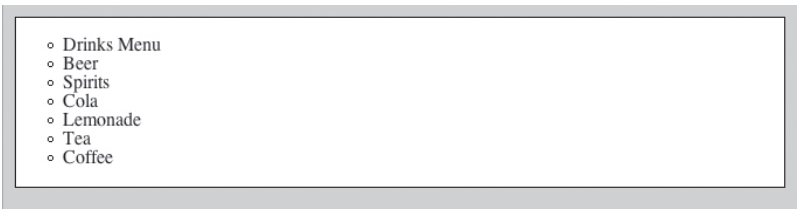
```
none
disc
circle
square
latin
```

Other possibly useful list-style-types may be required from time to time. Three useful examples include upper-alpha, lower-alpha, and upper-roman.

upper-alpha: A, B, C, D, E, etc.
lower-alpha: a, b, c, d, e, etc.
upper-roman: I, II, III, IV, V, etc.

Let's try some basic list styling. Create a new style sheet called lists.css, and remember to link to it in the head of lists.html. In lists.css you can try out each of the values by first adding the following selector for the unordered list element:

```
/* Styles for all default lists */
ul {
list-style-type:circle;
}
```



The bullets can be changed through CSS

Specifying disc would have made no difference as that marker was already used by default. Square would produce a small squared marker. If you wish to do away from any kind of basic marker use the 'none' value, which removes the marker altogether. You can also add a custom marker of your own using an image, as you'll discover later in the chapter.

6.1.2 Margin and padding

Notice that the actual list items do not move back to the left despite the lack of a marker, which leaves too much whitespace in front of each item. Therefore, the default padding applied by the browser can be reduced by specifying your preferred padding. Most browsers place the list items 30 pixels away from the left edge of the unordered list (default padding) and the unordered list itself approximately 10 pixels from the top edge of the container (default margin).

Notice also that each list item is block level. Therefore it will expand to fill the width of its container, and items above and below will wrap to a new line. Each of these defaults can be overridden with simple CSS declarations. To remove the default margin from the top and bottom of the unordered list, specify zero margins:

```
ul {
list-style-type:none;
margin:0;
}
```

The bullets are removed using the list-style-type: none value

To remove the default padding that pushes the listed items in by 30px, specify zero padding:

```
ul {
list-style-type:none;
margin:0;
padding:0;
}
```

To prevent the listed items from extending the full width of the container, set a maximum width:

```
ul {
list-style-type:none;
margin:0;
padding:0;
width:200px;
}
```

With default margin and padding settings removed, the unordered list and its contents now sit directly against the container edges. Therefore, if you will be using markers, but wish to override the default 30-pixel padding, you can specify a custom left padding value. In the following example, 20 pixels are enough to ensure the markers appear just inside the container. Note that it is specified using padding shorthand (top, right, bottom, left).

```
ul {
list-style-type:none;
margin:0;
padding:0 0 0 20px;
```

```
width:200px;
}
```

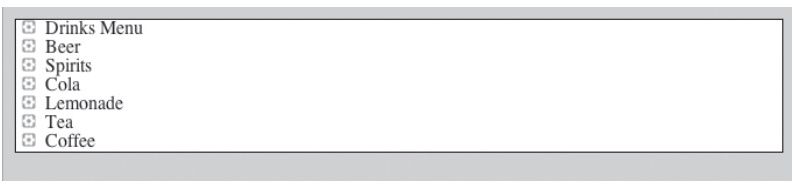
6.1.3 list-style properties

If you have created a bulleted list using a word processor you will be aware that as long lines of text wrap, they line up exactly to the left, with the bullet placed further left. The bullet is not treated as a character, and simply indicates where each new list item begins. This is how HTML lists also works by default. However, there may be occasions when the bullet needs to be inline with the text, and this is why the list-style-position property is used. The default value is outside, and though not specified in the preceding examples, it was already in operation. To override this default, inside can be specified (note that for this example, the default padding is also removed).

```
ul {
list-style-position:inside;
margin:0;
padding:0;
width:200px;
}
```

The typical bullet markers provided by CSS are fine for very simple lists, but in most cases a custom bullet marker will be more desirable. This accomplished using the list-style-image property, which allows a custom image to be used in place of the disc, circle, square, and other basic bullets. Now that you know how to space your list items away from the left edge of the unordered list, using custom images of any width shouldn't be a problem. For this example, we will use a small image with a size of about 12 pixels square, which have been named list.gif.

```
ul {
margin:0;
```



Here we have made use of a custom image as a marker

```
padding:0 0 0 25px;
width:200px;
list-style-image:url(images/list.gif);
}
```

As list-style-image is declared, there is no need to specify list-style-type. Note also that the left padding has been increased to compensate for the width of the image. The wider your custom bullet image, the more left padding you may need.

List properties can also be shortened into one declaration, using the list-style property. The order is `list-style-type`, `list-style-position`, `list-style-image`. `list-style: none inside url(images/list.gif);` However, it is not often that you would need to specify all three properties. In this example, declaring a custom image with `list-style-image: url(images/tile.gif)` removes the need to also turn off the basic list marker, except in the shorthand. Therefore, just specifying a custom image will be all you need in most scenarios.

Using the list-style-image property is the easiest method of assigning a custom bullet to your list items, but the results can be somewhat inconsistent. Some browsers will align the custom image directly in the middle of the list item text, while others will position it slightly higher. To beat this problem, a custom background image can be used for each list item. Note that default bulleting is turned off in the ul selector.

```
ul {
list-style:none;
```

Here the list markers are actually background images

```
}
li {
background: transparent url(images/list.gif) no-repeat
left center;
padding:0 0 0 25px;
```

```
}
```

For this example, the background-color is set to transparent in order to ensure that the white background colour of the container does not prevent the custom bullet images from displaying. Also, by using background-position values left center, the background image is forced to display the same distance from the top and bottom of the list element. Appropriate left padding is also declared to allow enough space in front of the list text for the image to fall into.

6.1.4 The inline list

By default, unordered list items will display vertically, with each list item on a new line. This is because the <li> element is a block-level element. There are, however, many occasions when you will need your list to display horizontally, for a main navigation bar, for example. This calls for the default block-level value to be overridden using the display property. Let's strip back the styling from the previous examples and examine the basic behaviour.

```
/* Styles for our basic list */
ul {
list-style-type:disc;
}
li {
display:inline;
}
```

Notice that the unordered list merely specifies the bullet marker type, and that it is the li selector that is now doing all the work. By giving display a value of inline; the default block display is overruled, forcing each list item to display on the same line. This is a great starting point for a horizontal menu, but what has happened to the bullet markers? The bullet will not show when display is set to inline. This brings us back to using background images with list items.

We have already seen how to apply a custom bullet as a background image in order to get around a few cross-browser display quirks. Using this approach allows you to assign a bullet for inline lists where default bullets refuse to be displayed. The CSS is fairly simple. Again, turn default bullets off in the ul selector, and ensure display:inline is declared in the li selector. Note also that the same background rules (transparent background and specific positioning) are declared.

```
ul {
list-style:none;
```

```
}
li {
display:inline;
background:transparent url(images/list.gif) no-repeat
left center;
margin:0 0 0 10px;
padding:0 0 0 15px;
}
```

The left margin and padding provide the space between them

As the list is now horizontal, everything appears on the same line and so the definition between each item is reduced. Therefore the left padding is reduced to just 15 pixels (just enough room for the 12-pixels-wide image), and 10 pixels of left margin are applied to space out each list item.

From this starting point, it should be clear how to further control the space between each item and get this thing to display exactly as you wish. In the next section, you'll learn how to take greater control of the list elements using good old IDs and classes.

6.1.5 Taking control with IDs

Let's give the drinks list its own unique ID. The approach to look at methods for controlling several lists and affecting the items they contain is based on this ID. In lists.html, add the following ID to the basic list markup, acting as a vital hook for many of the styles to be added throughout this section. Also, add a second list underneath that list (but still inside the container) with different list items and the ID food.

```
<ul id="drinks">
<li>Drinks Menu</li>
<li>Beer</li>
<li>Spirits</li>
<li>Cola</li>
<li>Lemonade</li>
<li>Tea</li>
<li>Coffee</li>
```

```
</ul>
<ul id="food">
<li>Food Menu</li>
<li>Toast</li>
<li>Cornflakes</li>
<li>Burgers</li>
<li>Steak</li>
<li>Salad</li>
<li>Fries</li>
</ul>
```

Each unordered list is given a unique ID. Thus, id="drinks" will not be used again on the page at any time, and allows that particular list to be styled uniquely. Likewise, the second list is the food menu, also unique. The CSS from the previous inline list example will suffice here, but if you need it again, use the following two selectors. Note that we are sticking with inline examples here, mainly to save page space, although everything is applicable to default vertical lists.

```
ul {
list-style:none;
}
li {
display:inline;
background:transparent url(images/list.gif) no-repeat
left center;
margin:0 0 0 10px;
padding:0 0 0 15px;
}
```

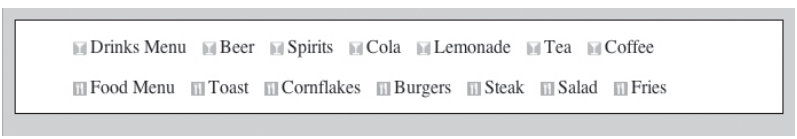
As things stand, the CSS will style both lists in an identical fashion, as it is not targeted to look for a particular ID

In some cases, this approach might be ideal, such as horizontal sidebar lists where all items should be rendered in the same way, but need to be separated into different unordered lists (to allow headings to be placed above each). However, the additional IDs added to each list allow you to target each with specific CSS selectors. For example, let's assign a different background image bullet for each list. As it is only the images that need to change, the ul selector is shared across both lists, and can be left unaltered. Also, all li declarations that are shared are first grouped together, before the targeted declarations are set separately.

```
/* Unordered list for drinks and food lists */
ul {
list-style:none;
}

li {
display:inline;
margin:0 0 0 10px;
padding:0 0 0 15px;
}

/* Images for drinks list only */
#drinks li {
background:transparent url(images/drinks.gif) no-repeat
left top;
}
/* Images for food list only */
#food li {
background:transparent url(images/food.gif) no-repeat
left center;
}
```



The food and drink components have separate icons

6.1.6 Grouping Items with Classes

In our two lists, there are clearly other groupings that can be defined. Drinks are grouped into alcohol, mixer, and hot by adding a matching class to each list element. In this example, the approach is used to group the food, using breakfast, dinner, and side classes.

```
<ul id="drinks">
    <li>Drinks Menu</li>
    <li class="alcohol">Beer</li>
    <li class="alcohol">Spirits</li>
```

```
    <li class="mixer">Cola</li>
    <li class="mixer">Lemonade</li>
    <li class="hot">Tea</li>
    <li class="hot">Coffee</li>
</ul>
<ul id="food">
    <li>Food Menu</li>
    <li class="breakfast">Toast</li>
    <li class="breakfast">Cornflakes</li>
    <li class="dinner">Burgers</li>
    <li class="dinner">Steak</li>
    <li class="side">Salad</li>
    <li class="side">Fries</li>
</ul>
```

Note that the first list item has no class attribute, yet every other item is assigned a class. This allows each drinks and food group to be treated individually. As before, the classes for each drink type are defined with unique shades of gray for font colour, but this time we can also group foods using the same method, by grouping the declarations:

```
/* Define first food and drink group */
.alcohol, .breakfast {
color: #333;
}
/* Define second food and drink group */
.mixer, .dinner {
color: #999;
}
/* Define third food and drink group */
.hot, .side {
color: #CCC;
}
```

The result sees the list of items move from black (the basic unordered list colour "drinks" and "food") through shades of gray (defined by the classes). Any further drinks or food added to the lists can be assigned to a particular drinks or food group, such as `<li class="side">Mashed Potato</li>`. Thus a logical colour key is established using simple CSS classes.

6.2 Nested lists

There will be many instances where you need to create a hierarchy in one list, and this is where nested lists are most useful. HTML allows you to begin an unordered list and create new lists inside that. The following markup takes the existing food and drinks menus and nests them inside one main list. It is very important to get the markup right in such scenarios, as to forget to close a top-level list item or nested list can cause all sorts of chaos.

```
<ul>
<li>Drinks Menu
<ul>
<li>Beer</li>
<li>Spirits</li>
<li>Cola</li>
<li>Lemonade</li>
<li>Tea</li>
<li>Coffee</li>
</ul>
</li>
<li>Food Menu
<ul>
<li>Toast</li>
<li>Cornflakes</li>
<li>Burgers</li>
<li>Steak</li>
<li>Salad</li>
<li>Fries</li>
</li>
```

```
</ul>
</ul>
```

Load it in the browser window, and the default browser style sheet will use a disc bullet for the top level and a circle bullet for the second level. Because a hierarchy is suggested with the markup, it is possible to make significant style changes without applying IDs or classes. First, let's use CSS to change the colour of the second level list items.

```
/* Style top-level unordered list and contents */
ul {
color:#000;
}
/* Style second-level unordered list and contents */
ul li ul {
color:#666;
}
```

Here the CSS selectors target the relevant list based on its relationship with another. Thus the first selector looks for the all <ul> elements it finds in the markup, and ensures the text is black, unless a declaration for a nested list says differently. Importantly, the second selector looks for any <ul> that is contained by one more above it, using the ul li ul descendant selector. In other words, the browser ignores the first <ul>, and any <li> elements inside it, but when it gets to the next <ul>, it will go to work and make its contents gray".

It becomes easy to define other custom values for each level. Building on our earlier example of assigning custom background images, the CSS can be used to target the second level of list elements specifically.

```
/* Style top-level unordered list and contents */
ul {
color:#000;
}
/* Style second-level unordered list and contents */
ul li ul {
list-style-type:none;
color:#666;
}
ul li ul li {
```

```
padding:0 0 0 15px;
background:transparent url(images/list.gif) no-repeat
left center;
}
```

So now the second-level ul selector (ul li ul) is still ensuring any information within it will be gray, but also turning off bullets for that list only. Also, the extra descendant selector for its list elements (ul li ul li) is tasked with applying a custom background image to those elements only.

6.3 Lists for navigation

A common feature of web sites built using web standards are navigation menus constructed using an unordered list. Using the `<ul>` element in this sense is semantically correct, with each destination link defined as an individual list item. This approach provides incredible flexibility, allowing the navigation list to be either horizontal or vertical as defined using CSS, and also allows for a seemingly unlimited number of styling approaches. Already in this chapter you have learned how to take the default vertical list and transform it into a simple horizontal navigation. Now, let's take things a step further and create a good-looking vertical navigation list, where each destination link is styled as though a graphic button.

6.3.1 The vertical navigation bar

A very common feature of many web sites is the vertical navigation bar created with simple list markup. The goal here is to turn each list element into a button without using any images whatsoever. Let's jump back to the concept of the simple list. For this task, a simpler unordered list is useful—just one level of list elements.

```
<ul>
<li>Beer</li>
<li>Spirits</li>
<li>Cola</li>
<li>Lemonade</li>
<li>Tea</li>
<li>Coffee</li>
</ul>
```

Remove All Default Spacing. This brings us back to the very simple display. First, the list elements need to be pulled to the top and left edges of the container.

```
ul {
list-style-type:none;
margin:0;
padding:0;
}
li {
padding:0;
}
```

This ensures you start with a blank canvas. No default spacing is in play. From this point on, all spacing is on your terms, avoiding the confusion that often comes with navigation list styling.

6.3.2 Turn list elements into buttons

Now some styling can be applied to make the list elements look a little more like buttons. The padding can now be adjusted also to provide enough space around the text.

```
ul {
list-style-type:none;
margin:0;
padding:0;
}
li {
background: #DDD;
margin: 0;
padding: 2px 10px;
border-left: 1px solid #fff;
border-top: 1px solid #fff;
border-right: 1px solid #666;
border-bottom: 1px solid #aaa;
}
```

In this example, shades of gray and white borders are used to give the buttons a slightly three-dimensional feel.

6.3.3 Define the width of the buttons

It is necessary to define a set width for the buttons, which can be done with the ul selector. Here 160px seems appropriate.

```
ul {
list-style-type:none;
margin:0;
padding:0;
width:160px;
}
```

6.3.4 Final touches

Now to refine the display a little more, font values are added with a shorthand declaration, and the unordered list is given a border, a margin, and 2 pixels of padding to create whitespace around the buttons.

```
ul {
list-style-type:none;
margin:5px;
padding:2px;
border:1px solid #333;
width:160px;
font: bold 12px 'Lucida Grande',Verdana,sans-serif;
}
li {
background: #DDD;
margin: 0;
padding: 2px 10px;
border-left: 1px solid #fff;
border-top: 1px solid #fff;
border-right: 1px solid #666;
border-bottom: 1px solid #aaa
}
```

This might be a good point to take what you have learned about navigation lists thus far and experiment with your own border, background, and margin values to personalize your list and get a feel for this vital element of CSS-based design.

6.4 The ordered list

The ordered list is a convenient way to mark up a list of items with each preceded by a number. HTML makes this possible with the <ol> element. Let's again take the drinks list and this time place it inside an ordered list.

```
<ol>
<li>Beer</li>
<li>Spirits</li>
<li>Cola</li>
<li>Lemonade</li>
<li>Tea</li>
<li>Coffee</li>
</ol>
```

The beauty of the ordered list is its flexibility. If it were necessary to add another drink to the list at any point, the automatic numbering would compensate appropriately, renumbering all list items that followed. Remember that other list-style-type declarations can be made to replace the auto generated numbers with other characters, such as upper-alpha, lower-alpha, and upper-roman.

6.4.1 Controlling the ordered list

While the default unordered list works like magic, there may be occasions where you wish to replace the default numerical characters with your own custom numbers. This is relatively simple, but does involve extra markup. The first thing to do is to identify each list item with its own unique class.

```
<ol>
<li class="one">Beer</li>
<li class="two">Spirits</li>
<li class="three">Cola</li>
<li class="four">Lemonade</li>
<li class="five">Tea</li>
<li class="six">Coffee</li>
</ol>
```

It should be noted that by utilizing this method, you are removing the browser's ability to automatically number your list items should you wish to insert another item in the middle. The reason an ordered list is useful here though is that if the style sheet were unavailable, the default numbers would be returned, and the list would still be ordered, keeping things semantically correct. The following approach will of course work for unordered lists, as well.

6.4.2 Creating custom numbers

The next step is to create the images you will use to replace the default text numbers. For this example, six images are needed. The size is the same as the custom bullets used earlier in the chapter (12×12 pixels)

6.4.3 Declaring the numbers using the unique classes

Notice that list-style-type:none is declared in the ol selector to turn off the default numbers (remember that this means no numbering will be displayed if the images are not available). Also, left padding is used to allow enough space for the custom number images. The padding needs only to be assigned to the li selector, as it is a value that is shared across all following selectors. Nothing new about that, but what is new is the six new selectors for each class that has been added to each list item.

```
ol {
list-style:none;
}
li {
padding:0 0 0 25px;
}
.one {
background: transparent url(images/ol1.gif) no-repeat left
center;
}
.two {
background: transparent url(images/ol2.gif) no-repeat left
center;
}
.three {
background: transparent url(images/ol3.gif) no-repeat left
center;
}
.four {
background: transparent url(images/ol4.gif) no-repeat left
center;
}
.five {
background: transparent url(images/ol5.gif) no-repeat left
center;
}
```

```
.six {
background: transparent url(images/ol6.gif) no-repeat left
center;
}
```

So you now have six selectors that correlate with the classes assigned to the ordered list. Mixing all of this together and loading the page in the browser gives the basic customised list.

6.4.4 Dressing up the ordered list

With the custom icons working, you are now free to add further styling to the ordered list. Many of the declarations used to create the vertical navigation list earlier are reused here, specifically the background colours and borders.

Note, however, that the unique classes added to each list item overrule any background-color declaration in the li selector, so background-color needs to be declared for each class. The padding is also adjusted to replicate that in the vertical navigation example, allowing the custom numbers and list item text to be spaced appropriately.

```
ol {
list-style-type:none;
margin:5px;
padding:2px;
border:1px solid #333;
width:160px;
font: bold 12px 'Lucida Grande',Verdana,sans-serif;
}
li {
margin: 0;
padding: 2px 10px 2px 25px;
border-left: 1px solid #fff;
border-top: 1px solid #fff;
border-right: 1px solid #aaa;
border-bottom: 1px solid #666
}
.one {
background:#DDDDDD  url(images/ol1.gif)  no-repeat  3px
center;
}
```

Figure 7-9: The finished navigation bar

```
.two {
background:#DDDDDD url(images/ol2.gif) no-repeat 3px center;
}
.three {
background:#DDDDDD url(images/ol3.gif) no-repeat 3px center;
}
.four {
background:#DDDDDD url(images/ol4.gif) no-repeat 3px center;
}
.five {
background:#DDDDDD url(images/ol5.gif) no-repeat 3px center;
}
.six {
background:#DDDDDD url(images/ol6.gif) no-repeat 3px center;
}
```

One further change is made to the shorthand background-position property for each custom number. Instead of left, a value of 3px is specified to pull the custom image away from the left border of the list item. The end result looks very similar to the earlier vertical navigation, except the custom numbers are in place.

The drawbacks of this technique are obvious, but the benefits are far-reaching. One drawback is that this method works well only for ordered lists that are unlikely to grow or be reordered, as you are now overruling the default browser action of reordering and number adjustment. A major positive is that you can now see how easy it is to take greater control of any kind of list, as the method of applying unique classes to list elements will of course work for unordered lists also.

7 Links

The most important components of a web page are undoubtedly its links. It is very rare that a web page doesn't link to another. A Google results page is all about links. Creating a link isn't difficult, but managing to link to another web page is only the beginning. With CSS, links can be made both beautiful and functional. From rollover colours to advanced image maps and complex menus, link styling is a vast and evolving area of web design.

Whenever you create a link you have to consider the end user, but a link has a very important role to play, and its basic functionality should never be compromised in pursuit enhancement. There are a million things you can do with links to enhance the user's experience. Let us check out how this can be done.

7.1 Link markup

The link is a very simple concept, whether you are using text or an image it's one of the first bits of code that most of us get to know when dealing with HTML. The basic markup for a text link is as follows:

```
<a href="http://www.google.com">Google</a>
```

Or to use an image as the link and ensuring that the alt attribute is used to provide relevant link text should the image not be available:

```
<a href="http://www.google.com"><img src="images/google.gif" alt="Visit Google" /></a>
```

Other attributes are also available, such as title, used to provide a tool tip for additional information when the user hovers over the link:

```
<a href="http://www.google.com" title="Visit the best search engine in the world">Google</a>
```

This markup is all that is needed to hook up CSS rules for styling the links. Naturally, as links are self-contained elements, they can be combined with parent elements such as lists, or housed in custom containers in order for more specific descendant selectors to take effect and treat links in different ways depending upon where they sit in the page.

7.2 Default link styling

By default, browsers have a set CSS for links. If you have ever worked with a link without applied CSS you will be aware that the default browser style sheet will render unvisited links in dark blue, visited links in purple, and so on, and that these links will be underlined. This convention is a staple

approach to the Web since its early days and is easily recognizable for quickly analysing which links on a page the user has already visited

7.3 Template HTML markup

Let's start learning the rules by first setting up a new template, and then approach some simple CSS link declarations. Create a new template called `links.html` and paste the following markup into it:

```
<h2>Introducing the band</h2>
<p>To find out more about members of <a href="#">The Dead
Goods</a>, please select the appropriate person for a full
profile.</p>
<ul>
<li><a href="links.html"> </a></li>
<li><a href="#">John Lennon</a></li>
<li><a href="#">Jimi Hendrix</a></li>
<li><a href="#">Jeff Buckley</a></li>
<li><a href="#">Kurt Cobain</a></li>
<li><a href="#">Janis Joplin</a></li>
<li><a href="#">Keith Moon</a></li>
</ul>
```

Notice that the link to # links to the actual template created. This allows a visited link state to be available at all times which is essential for checking the visited link styling to be added shortly. Notice also that one link is placed in the opening paragraph, and the rest are correctly defined as list items. Save this template and load it in your browser. Notice that all links are the default dark blue, aside for the link to Judas Priest, which links to the page in view and so is a visited link, displayed in purple. Next, link to an external style sheet called `links.css`. Add declarations for the body selector, and also for the container div if you are using one.

7.4 Changing Link Colour (The different CSS states for a link)

Now let us add some simple CSS rules to take control of the links. This is done with the pseudo classes, :link, :visited, :active, and :hover. First, the pseudo class: link is combined with the a element selector (from the <a href... part of the link markup), creating the selector a:link.

```
a:link {
```

```
color:#F00;
}
```

This simple selector targets all instances of the <a> element and turns all unvisited links to red. Any visited links will still be purple, as thus far you have not created a selector to override the browser's default visited link style. To show visited links, simply create a selector for the <a> element with the :visited pseudo class.

```
a:visited {
color:#999;
}
```

Now all unvisited links are red, and all visited links will be light gray. It is very important to make visited links appear different from unvisited ones. It is an accepted convention and instantly highlights any links already followed on that machine.

If you are going to have a vast list of links, you need some form of interaction. The user could easily click the wrong link and be taken to the incorrect destination. The answer to this problem is the powerful :hover pseudo class, which aside from being a great usability aid can also make the links attractive. Again, the selector first looks for all instances of the <a> element, but will only be performed when the user's mouse pointer hovers over the link text.

```
a:hover {
color:#333;
}
```

This rule will turn any link text dark gray on mouseover without exception, whether the link is visited or unvisited. The :active pseudo class takes care of the link styles when the mouse button is actually clicked, and is another useful usability aid.

```
a:active {
color: #000;
}
```

As the user clicks the link, the text will turn black for as long as the current page remains in view, acting as an extra clue to show the user where he or she is about to be taken.After creating your pseudo classes in the order shown previously, you will have the following selectors in the links.css style sheet:

```
a:link {
color:#F00;
}
```

```
a:visited {
color:#999;
}
a:hover {
color:#333;
}
a:active {
color:#000;
}
```

Notice that the first letter of each pseudo class is in bold, giving us the letters L V H A. This order is very important of you wish for the links to behave properly, and can be remembered as LoVe HAte. For example, if you were to place the a:hover selector above the a:visited selector (giving you the order L H V A), you would have a situation where the a:hover declaration would have no effect upon the visited state because of the way the cascade works. The text would remain light gray even on hover, going against expected behaviour. There may be cases where such an approach is required, but this will be rare and is probably best avoided.

Now that you understand the basics of CSS link control and can affect any of the four core link states, it is worth considering some other very useful properties designed specifically for links, and some more universal properties that can also be applied.

7.5 Basic CSS rules for links

7.5.1 text-decoration

So far, you may have noticed that all links, regardless of state, are underlined with a colour matching the link text. This is a default style that is very easily removed using the text-decoration property. Possible values are none, underline (the default value), overline, line-through, and blink. These values are very self-explanatory, but again be very aware of how usable and intuitive your links are when custom styles are used. The text-decoration property can be applied to any of the four link states, as in this example for the :hover pseudo class:

```
a:hover {
color:#333;
text-decoration:none;
}
```

This is a fairly sensible approach, as on hover, the link colour will change to satisfactorily identify the link, so underlines are not needed. It is generally recommended that all other link states carry an underline to distinguish them from normal inactive text, and it is not a good idea to identify links with colour alone.

7.5.2 Using borders with links

The border property turns off the default underlines and replaces them with custom bottom borders and can be used to create some good effects. In the example, all basic links will be given a thin dotted underline, which is actually a 1-pixel bottom border:

```
a:link {
color:#F00;
text-decoration:none;
border-bottom:1px dashed #333;
line-height:150%;
}
```

Note that unlike the text-decoration property, here the border can be a different colour from the link text, so you can have red text with dark gray underlines. Also, with a line-height declared, you ensure that the underline doesn't encroach upon any text underneath it.

7.5.3 Adding symbols with background images

It should also be obvious that a background image can also be applied to links. Here a different background image can be applied to each of the four link states, further aiding usability by providing a visual indicator alongside the more traditional text variations. For this example, the goal is to use an arrow symbol to indicate links not yet visited and a checkmark to indicate links already visited. For this you will need two small GIF images sized approximately 12×12 pixels.

As each image is 12 pixels in width, it makes sense to provide enough padding to the right of the link text for the image to be placed. Here, 15 pixels are declared for the right padding. Also, the arrow image is assigned to the background property for the :link pseudo class, set to only appear once, at right center. The checkmark image is assigned in an identical way for the :visited pseudo link.

```
a:link {
color:#F00;
padding:0 15px 0 0;
background:url(images/arrow.gif) no-repeat right center;
```

```
}
a:visited {
color:#999;
padding:0 15px 0 0;
background:url(images/checkmark.gif)  no-repeat  right
center;
}
```

And so all unvisited links appear with a small arrow symbol to their immediate right, and any visited links have a neat checkmark in place of the arrow. In this example, only two link states have been affected. As things stand, both the active and hover states will place the arrow symbol to the right of the link text, but there is nothing to stop you defining a custom symbol for those states also.

7.6 Targeting Links with Descendant Selectors

In the previous section, the four pseudo classes were used to apply CSS rules to all links wherever they appear on the page. There will undoubtedly be situations where you need a specific link treatment for a specific section of the page (such as the footer, main navigation, or a sidebar). Thankfully, this is easily achieved with a few more selectors. In the example template links.html, most of the links are contained in an unordered list, but one is within the first paragraph. As the latter is contained in a separate parent element, it can be targeted with a descendant selector and treated differently. The four pseudo classes can be individually rewritten for any links inside paragraphs, by again assigning the colour for each state, for instance. However, it is likely that some of the default link styles will still be needed. In this example, existing colour declarations assigned in the existing pseudo classes (red for unvisited, gray for visited, and so on) are to be retained. Here are the defaults that control all links anywhere on the page:

```
a:link {
color:#F00;
}
a:visited {
color:#999;
}
a:hover {
color:#333;
text-decoration:none;
```

```
}
a:active {
color: #000;
}
```

These default pseudo classes can be left alone; as all that is needed is a new descendant selector that applies extra declarations to these existing classes.

```
p a:link, p a:visited, p a:hover, p a:active {
color:#F00;
padding:2px;
border:1px dashed #999;
text-decoration:none;
}
```

Here, the browser will look first for a paragraph, and then any link elements contained within. If one or more is found, the new declarations will be added to the existing pseudo classes. So, in addition to the colour declaration in the :link pseudo class, the new declarations (padding, border, and text-decoration) will be applied also.

Therefore, the same treatment can be given to links housed within another element, such as the list of links in the unordered list. To revisit the background images example from earlier in the chapter, the links in the list could be targeted as follows:

```
ul a:link {
color:#F00;
padding:0 15px 0 0;
background:url(images/arrow.gif) no-repeat right center;
}

ul a:visited {
color:#999;
padding:0 15px 0 0;
background:url(images/checkmark.gif)  no-repeat  right
center;
}
```

7.7 Transforming a navigation bar with links

We've seen how CSS can be used to transform a simple unordered list into a vertical navigation bar. It is useless if it doesn't actually link anywhere, so as discussed it is now time to make that simple list work as a navigation tool.

This example takes the list styling from one of our previous examples and reworks it for a list of links. Create a new template called `linkslist.html`. Again, note that in the examples the following is placed inside the body element, and also inside a container div as in previous chapters.

```
<ul>
<li><a href="#">Beer</a></li>
<li><a href="#">Spirits</a></li>
<li><a href="#">Cola</a></li>
<li><a href="#">Lemonade</a></li>
<li><a href="#">Tea</a></li>
<li><a href="#">Coffee</a></li>
</ul>
```

Now create the `linkslist.css` file using the version provided previously. Remove any existing declarations for `<ul>` and `<li>` elements, and ensure the following declarations are added alongside existing declarations for `<body>` and container:

```
ul {
list-style-type:none;
margin:5px;
padding:2px;
border:1px solid #333;
width:160px;
font: bold 12px 'Lucida Grande',Verdana,sans-serif;
}
li {
background: #DDDDDD;
margin: 0;
padding: 2px 10px;
border-left: 1px solid #fff;
border-top: 1px solid #fff;
border-right: 1px solid #aaa;
border-bottom: 1px solid #666
}
```

This will display the unordered list almost exactly as it did in the previous chapter, except that the list items are now links, and as there are no pseudo class declarations in the style sheet, the links display in the default blue and

are underlined.

The next step is to define any shared declarations for links within the unordered list. For this example, let's turn off all underlines. If you decide that perhaps you'd like underlines on the hover state, simply remove the ul a:hover selector from the grouping. Very importantly, the padding declaration must be removed from the li selector and placed instead in the grouped values for the pseudo classes. This is because the whole list element needs to be a clickable link, so by adding that padding in the pseudo classes, the active area is not merely the link text, but is increased to the whole link element. Finally here, display:block is added. This ensures that the active link area is the whole width of the link element. This means that one can follow the link by clicking anywhere in the list element, not just by clicking the link text.

```
ul a:link, ul a:visited, ul a:hover, ul a:active {
display:block;
padding:2px 10px;
text-decoration:none;
}
```

Next, unique selectors can be added to assign specific declarations for each link state. Note that again the selector descends through ul and a before the pseudo class is stated, ensuring the rules are targeted and will not affect links elsewhere in the page.

Visual changes can be observed when clicking the links

```
ul a:link {
color:#000;
}

ul a:visited {
color:#666;
}
```

```
ul a:hover {
color:#F00;
}
ul a:active {
color:#333;
}
```

So far, the link text is changing depending on the link state, but you can also change the colour of the whole link background on hover. This is simply

Link States (a) Link, (b) Visited, (c) Hover

done by declaring a background colour in the appropriate pseudo class. In the following CSS, the hover pseudo class is given a white background:

```
ul a:hover {
color:#F00;
background:#FFF;
}
```

As a result, when the mouse is placed over a link, the text turns red, and the whole background of the list item becomes white. Obviously, you could also define a background colour for visited links or any pseudo classes.

The final trick is to highlight a link if you are actually viewing that page. For static web pages, this requires a particular link to be identified by adding a unique ID to the parent list element. With dynamic systems such as content management systems and languages such as PHP, the unique ID could be added in a more convenient fashion by testing for the current page based on the URL or other identifier, and placing the ID accordingly.

Let's say that for the Lemonade page, you wish to reflect that the user is actually viewing that page, and therefore make it obvious that clicking that link again would be pointless. All that is needed is for the unique ID to be added to that particular list element.

```
<ul>
<li><a href="#">Beer</a></li>
<li><a href="#">Spirits</a></li>
<li><a href="#">Cola</a></li>
<li id="current"><a href="#">Lemonade</a></li>
<li><a href="#">Tea</a></li>
<li><a href="#">Coffee</a></li>
</ul>
```

A new CSS selector is required to declare the appropriate styles for the current ID. Because the ID is combined with the :link pseudo class, any values declared in this selector will override those specified in the link pseudo classes.

```
#current a:link {
color:#FFF;
background:#333;
}
```

The key to great link styling is awareness of the end user. Make hyperlinks

By using a unique ID the current page can be highlighted

are accessible and intuitive. Don't confuse people by employing some ridiculous colour scheme that has no hierarchy. Think carefully about the contrast between colour and background, and consider font and font size. Be creative, and by all means be inventive, but above all, be careful. If there is one thing that frustrates a user above all else, it must be badly thought-out navigation and ill-conceived link treatment.

8 Layout tools

Now that we have the basics of styling the core elements of a web page, let's get to a broader perspective, designing the page layouts and the tools needed for it such as floats and positioning. By exploring how they can be applied to basic elements with a core understanding of these key concepts, you'll be armed and ready to face all aspects of CSS layout.

8.1 Floats

The concept of floats is instrumental in designing a layout using CSS. Floats allow you to work against the linear nature of the flow of elements on a normal HTML page. Without floats, each element will get placed below the one above. There would be no columns and no inline images. Before CSS, this problem was overcome to a certain extent by making use of tables in HTML while designing a web page layout.

When you float an element, it becomes a block-level element that can then be shifted to the left or right on the current line. A floated box is laid out according to the normal flow of elements, but it's then taken out of the flow and shifted to the left or right as far as the containing element will allow. Content such as text can flow down the right side of a left-floated box and down the left side of a right-floated box. Floats are needed for placing images between text, creating columns, and making any design with more than one element horizontally.

But understanding floats is really important when you get to using them, because you might find a few pitfalls along the way. Elements following a floated element will wrap around it. If you do not want this to occur, those following elements need to be "cleared," essentially reverting back to the natural flow of page elements. Floats are also required to ensure containing elements do their job and actually contain any floated elements within. Failing to clear those floated elements can result in a container collapsing long before the end of the child elements it contains. Three possible values are available for the float property: 'left', 'right' and 'none'

Gaining a core understanding of floats, is necessary for designing a multi columned layout which is very common these days. Aligning a small image next to a block of corresponding text does not happen normally, you add the image into the HTML and it will always show up only below the text. But when you need a thumbnail image inline with basic introduction text for an article, you have to float the image.

Place the following (X)HTML in a new file called floats.html:

```
<html>
<head>
<title>Chapter 10: Layout Basics</title>
<link rel='stylesheet' media="screen" type='text/css'
href='float.css' />
</head>
<body
<div id="container">
<h2>Introducing the band</h2>
</div>
</body>
</html>
```

Save the file. Note that an external style sheet called float.css is referenced in the <head> element of the document. Create float.css and paste the following code into it:

```
/* Specify blanket rules for all elements */
body {
font-size:80%;
font-family:'Lucida Grande',Verdana,sans-serif;
margin:10px;
background-color:#CCC;
}
/* Container for all page content */
#container {
padding:10px;
border:1px solid #000;
background-color:#FFF;
}
/* Rules for headings */
h1 {
font-size:150%;
}
h2 {
font-size:140%;
}
h3 {
font-size:120%;
```

```
}
p {
font-size:100%;
line-height:150%;
}
```

Save float.css and load the HTML file in your browser. Next, create a small thumbnail image, roughly 80×80 pixels, and place it in your images directory. Now copy and paste the following into the content area div of the file, being sure to add the appropriate path to your image.

```
<h3>Fast Track</h3>
<img src="/images/priest.gif " />
<p>Can't actually play any instruments.</p>
```

Save and load the file. In itself this is not an ugly layout, but with the image sitting on its own line, it is taking up valuable page real estate. Now let's float the image. First, define the image element by applying a simple `<div>` element as its parent.

```
<h3> Fast Track </h3>
<div class="image_float"><img src="/images/priest.gif" </
div>
<p>Can't actually play any instruments, and is just along
for the ride supposedly as manager.</p>
```

With this in place, the image can now be easily manipulated using a simple float. Note that you could also float the image by declaring `image_float` as a class within the `<img>` element.

```
<h3> Fast Track </h3>
<img src="/images/priest.gif" class="image_float" />
<p>Can't actually play any instruments, and is just along
for the ride supposedly as manager.</p>
```

As previously noted, the float property can be given values of left, right, or none (the latter useful to override any blanket floats where an exception is needed). Let's first float the image left. Add the following chunk of CSS to the float.css style sheet and save the file:

```
.image_float {
float:left;
```

```
margin:0 5px 5px 0;
}
```

Here, as well as define the float, right and bottom margins have been declared using shorthand to ensure the text surrounding the image will not sit against its edges. The float should work and in that the image is indeed floating to the left, and the paragraph text is flowing around it to the right, spaced away nicely by the declared margin. The problem, however, is that the containing (container) doesn't recognise the float, and as a result it is not expanding vertically to contain the <image_float> element, only the now-less-tall paragraph. It must therefore be a good time to look at clearing floats.

8.2 Clearing floats

Elements following a floated element will wrap around that floated element. For example, if an image floats left and is followed by paragraph text, that text will wrap around the image and continue directly underneath it if long enough. If you do not wish this to happen, you can apply the clear property to those elements that follow the float. Four options are available for the clear property: 'left', 'right', 'both' and 'none'

8.2.1 clear:left

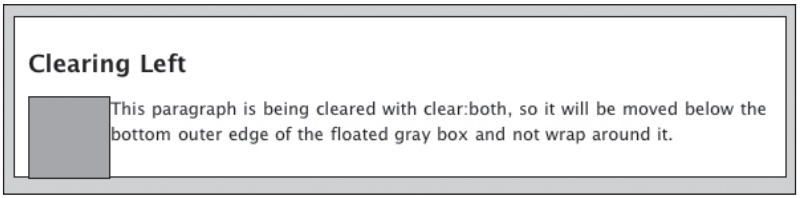
By specifying clear:left rule, the element moves below the bottom outer edge of any left-floated elements. Consider the following HTML.

```
<h2>Clearing left</h2>
<div class="floatbox"></div>
<p>This paragraph is being cleared left, so it will be
moved below the bottom outer edge of the floated gray
box.</p>
```

Some simple CSS selectors are used first to define paragraph properties and second a simple gray box that will be floated left.

```
p {
font-size:100%;
line-height:150%;
}

.floatbox {
float:left;
```



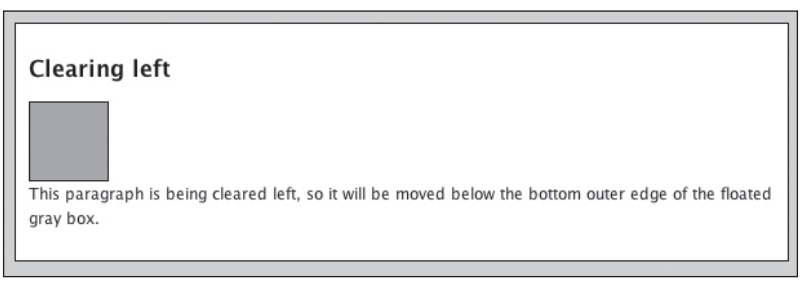
If the paragraph is not cleared the text will flush right against the gray box

```
width:60px;
height:60px;
background-color:#999;
border:1px solid #000;
}
```

The text is lying, as the text is blatantly still wrapping around the box. So, we need the paragraph to clear the floated gray box. To do this, the clear property is added to the paragraph selector, as follows:

```
p {
clear:left;
font-size:100%;
line-height:150%;
}

.floatbox {
float:left;
width:60px;
height:60px;
background-color:#999;
```



The gray box is floated left

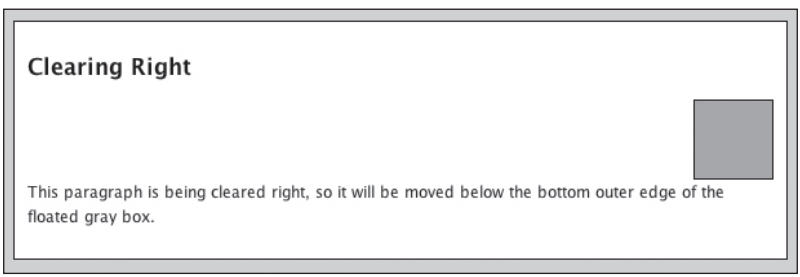
```
border:1px solid #000;
}
```

The box is still floated left, and that the paragraph is now cleared left. This will force the paragraph to begin on a new line underneath the box .

8.2.2 clear:right

Specifying clear:right will ensure that the element is moved below the bottom outer edge of any right-floated elements directly above. Let's examine another example built upon the previous one. In the CSS, all that is needed is to change the instances of left to right.

```
p {
clear:right;
font-size:100%;
line-height:150%;
}
```



The gray box is floated right

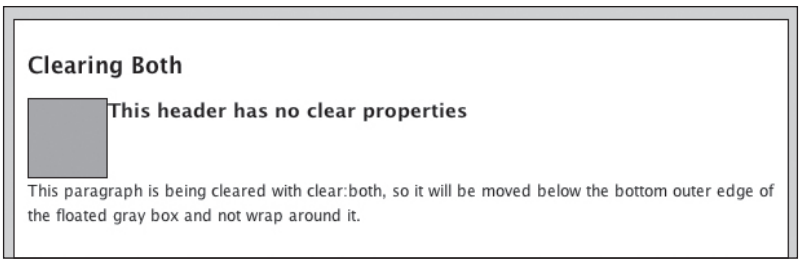
```
.floatbox {
float:right;
width:60px;
height:60px;
background-color:#999;
border:1px solid #000;
}
```

Failing to change the paragraph's clear value from left to right would

result in the paragraph text appearing to the left of the floated box, and on the same line.

8.2.3 clear:both

By specifying clear:both, the element is moved below all floating elements, regardless of whether they are floated left or right. In this example, the box is floated left, but the clear property for the paragraph is given the both value.



The gray box is floated left

```
p {
clear:both;
font-size:100%;
line-height:150%;
}
```

In the HTML, a simple level 3 heading is added after the floated box. This heading has no float or clear properties.

8.2.4 Clearing your floated image

Remember that although the image is being floated left, and the text is flowing around it correctly, the containing element is not recognising the float, and is ending after the paragraph, and not the taller image. Clearing the float can now solve this problem, but the solution is less than perfect, in that it calls for an extraneous element to be applied.

Were there another element following the paragraph (such as another level 3 heading), the clear property could be defined for that element, removing the need for a special element to do the clearing. We'll do this later, but for now, there is no following element, so the extraneous spacer `<div>` is needed.

Immediately after the <p> element, add the spacer element .This horrible element is unquestionably presentational, but unavoidably necessary to force the container to stretch vertically to hold the image. The next step is to define CSS rules for the spacer:

```
.spacer {
clear:left;
}
```

By giving clear a value of left, the spacer is moved below the bottom outer edge of both the image float element and the paragraph. Although the spacer is invisible in the final result, it is now forcing the container to expand vertically to contain it.

Clearing the float is necessary in most similar situations, as it may be unknown as to how much paragraph text is to flow around the floated element. If more paragraph text were added so that it extends beyond the height of the floated image, the spacer would not be necessary, as the container would expand to contain the paragraph. Still, to be on the safe side, it is worth adding the spacer should the text be decreased at a later date.

8.3 Handling multiple floats

It makes sense to add more profiles underneath, within the main container. Without the use of spacer s, things aren't quite right. The level 3 heading is correctly wrapping around my floated image. The first approach here might be to add a spacer after each profile, as follows:

```
<h3>Judas Priest</h3>
<div class="image_float"><img src="/images/priest.gif"
alt=" Judas Priest
mugshot" /></div>
<p>Can't actually play any instruments, and is just along
for the ride supposedly as manager.</p>
<div class="spacer"></div>
<h3>Jimi Hendrix</h3>
<div class="image_float"><img src="/images/hendrix.gif"
alt="Jimi Hendrix's mugshot" /></div>
<p>Possibly the greatest guitarist who ever walked the
earth, Mr. Hendrix is thankfully alive once more to play
the next <em>Dead Goods</em> tour across the North of
England.</p>
<div class="spacer"></div>
```

Now the layout is using two instances of the extraneous presentational spacer element, adding unnecessary markup bloat. There are seven people in this band, so that means by the end of the profile section, there will be seven instances of the spacer element. There is a better way to do things

8.4 Clearing with existing elements

Earlier we discussed if there were another element following the paragraph (such as another level 3 heading), the clear property could be defined for that element, avoiding the need for a spacer . Well, as there are now more profiles, there are now other elements following each block of profile information. Here, a level 3 heading begins each profile, so this will be the element used to clear the float above. Focusing on the markup inside the container, let's add a couple more profiles to the HTML. Note that all but the very last spacer is removed.

```
<h2>Introducing the band</h2>
<h3>Judas Priest </h3>
<div  class="image_float"><img  src="/images/priest.gif"
/></div>
<p>Can't actually play any instruments, and is just along
for the ride, supposedly as manager. </p>
<h3>Jimi Hendrix</h3>
<div  class="image_float"><img  src="/images/hendrix.gif"
alt="Jimi Hendrix's mugshot" /></div>
<p>Possibly the greatest guitarist who ever walked the
earth, Mr. Hendrix
is thankfully alive once more to play the next <em>Dead
Goods</em> tour across
the North of England.</p>
<h3>Janis Joplin</h3>
<div  class="image_float"><img  src="/images/joplin.gif"
alt="Joplin's
mugshot" /></div>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Vestibulum in lacus. ... Quisque vitae lorem
placerat risus posuere congue.
Integer a orci.</p>
<h3>John Lennon</h3>
<div  class="image_float"><img  src="/images/lennon.gif"
```

```
alt="Lennon's mugshot" /></div>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Vestibulum in lacus. ... Quisque vitae lorem
placerat risus posuere congue. Integer a orci.</p>
<div class="spacer"></div>
```

That last spacer will still be required, as the final profile has no elements following it, but still needs to be cleared to force the container to stretch around it. Now the level 3 heading can be used to clear the floats that precede it. The same clear value used in the spacer is applied to the selector for h3:

```
h3 {
clear:left;
padding-top:20px;
font-size:120%;
}
```

Note that padding-top is also declared to create more space between each profile and make things a bit prettier. Now, the level 3 heading is doing all the work, clearing the preceding float and ensuring each profile begins on a new line. It is just as simple to float images to the right, and some designers prefer to use this approach, as it generally looks neater. The lines of text that wrap unevenly (unless lines of text are given equal length using text-align:justified, which has very unpredictable results) to create a jagged right edge can be neatened up nicely by placing the image to the right. The first step is to simply adjust the value of the float to right, and to define margins to the left of the image instead of the right.

```
.image_float {
float:right;
margin:0 0 5px 5px;
}
```

Thus, a simple adjustment to the two selectors used to clear floats is required. Before we were clearing left, we now need to simply clear right.

```
h3 {
clear:right;
padding-top:20px;
```

```
font-size:120%;
}
.spacer {
clear:right;
}
```

These approaches to floats merely skim the surface of possibilities, and by applying floats to things like dates, icons, and links, some very neat, space-saving effects can be achieved. Importantly, these basics will also form the crux of our approach to column layout.

8.5 Positioning

Positioning allows designers to place HTML elements with a greater accuracy using some simple CSS rules. The position property determines the reference point for the positioning of each element box. All boxes start out being positioned in the normal flow of elements in the document. CSS positioning is a difficult topic to grasp.

Let's begin by defining the four main types of positioning as quickly and simply as possible, before looking at each in more detail:

8.5.1 Static positioning

The easiest to understand, and closest to what you have already been doing in this book. This term basically describes how elements are placed by default. The browser takes your HTML and parses it into the individual elements, applying CSS to each as directed, and finally collating all of this into the visible web page. The final position an element takes as a result of this is its static position, and as such, it is nothing particularly special.

8.5.2 Absolute positioning

Absolute positioning allows you to dictate where the top-left corner, bottom-right corner, or other point of reference of an element will sit in relation to the nearest parent element that has been positioned out of the flow of the document. When the web page is scrolled, the elements retain their position against each other, and all scroll with the page as though glued together.

8.5.3 Fixed positioning

Fixed positioning allows an element to be placed in relation to the actual browser window. Therefore, if the page is scrolled, everything moves with it,

except the fixed element, which holds its ground x pixels from the top, left, right, or bottom of the browser window.

8.5.4 Relative positioning

A relatively positioned element is relative to where it would normally be positioned statically. For example, if applied to a heading, you are defining where it should be relative to where the browser would normally put it.

8.5.5 Basic position properties and values

There are many possible properties to be used when positioning elements. For now, we will look into the following properties:

```
position
top
left
bottom
right
```

Other position-related properties will come into play later in the book, especially when you come to learning about layouts. For now, let's look at how position and associated properties affect basic elements.

Once the position of the element is defined, the top, left, bottom, and/or right properties are used to offset that element by the given values. All values can be defined using length, percentage, or auto. As stated, the offset is entirely dependent upon the value of the position property.

By applying CSS to the HTML example, it is easy to see how each position value works with the same top and left values. Note that for our position examples, the following HTML is used and the aim will be to control the position of the image (<img>) element.

```
<div id="container">
<h3> Judas Priest </h3>
<div class="image_float"><img src="/images/priest.gif"
alt=" Judas Priest mugshot" /></div>
<p>Can't actually play any instruments, and is just along
for the ride, supposedly as manager..</p>
</div>
```

The container div is given specific width and height values so that the browser window can be forced to scroll in order to see the effects of all position values.

```
/* Define the container for positioning examples */
#container {
width:400px;
height:400px;
margin:10px;
padding:10px;
border:1px solid #000;
background-color: #FFF;
}
```

8.5.6 absolute

The element is placed to the top, right, bottom, left, or combination thereof of the nearest parent element that has been placed out of the document flow.

8.5.7 fixed

The element is placed to the top, right, bottom, and/or left of the browser window.

8.5.8 relative

The element is placed to the top, right, bottom, and/or left of the element's default position.

```
p {
font-size:100%;
line-height:150%;
}
```

Here, the position relates not to any parent element or to the browser window, but specifically to the properties of the element itself. Notice that the gap remains between the heading and the paragraph, where the image would normally sit in its default position. Let's move the image again. Here, the position is first defined as absolute, and then the coordinates are set using top and right.

```
/* Define image position */
img {
position:absolute;
left:400px;
}
```

By declaring left as 400px respectively, and combining this with position:absolute, the image is moved appropriately in relation to the

viewport. Note that the paragraph has shifted up, and that there is no gap where the image would be naturally. Also, while it might appear that the image is positioned in relation to the browser window, it isn't fixed, and if scrolled, the elements all move together as if glued into place. Thus, the image is absolutely positioned.

```
/* Container for all page content */
#container {
position:relative;
top:100px;
width: 400px;
height: 400px;
margin: 10px;
padding:10px;
border: 1px solid #000;
background-color:#FFF;
}
/* Define image position */
img {
position:absolute;
left:400px;
}
```

Here, the container is moved 100px from where it would normally sit, using position:relative and top:100px. So, because the image inside is absolutely positioned, it will take its starting point not from the viewport, but from its original position within the container.

```
/* Container for all page content */
#container {
width: 400px;
height: 400px;
margin: 10px;
padding:10px;
border: 1px solid #000;
background-color:#FFF;
}
/* Define image position */
```

```
img {
position:fixed;
left:400px;
}
```

As a result of this change, the image's position will appear to be the same as in the absolutely positioned example. The real surprise comes when you scroll the .The image remains in view always, fixed 400px from the left of the browser window. This technique is perfect for navigation elements or forms that need to appear in view no matter how far the user scrolls down the web page.

9 Types of Layouts

After learning about float and clear, designing CSS based layouts should be very easy. A key factor in choosing a layout is the audience. Not every visitor to your web site will have the same browser. They will not share an identical screen resolution, but will definitely have their browser windows at varying widths. You have to begin to design for the lowest common denominator. We will examine some of the most common approaches to CSS layout with particular emphasis on flexibility and cross-browser performance. Deciding which layout is best for your web site is entirely up to you.

We'll investigate something called the Box Model, with specific regard to margin and padding, and how to get your columns to behave when viewed with older browsers. The various types of layouts include fixed width, liquid, elastic, and variable fixed width, for starters. Then there are other decisions to be made: should there be two, three, or four columns, and should the columns be floated or positioned? We will examine just a few of the possible options available to get a basic idea.

9.1 Different types of layouts

9.1.1 Fixed

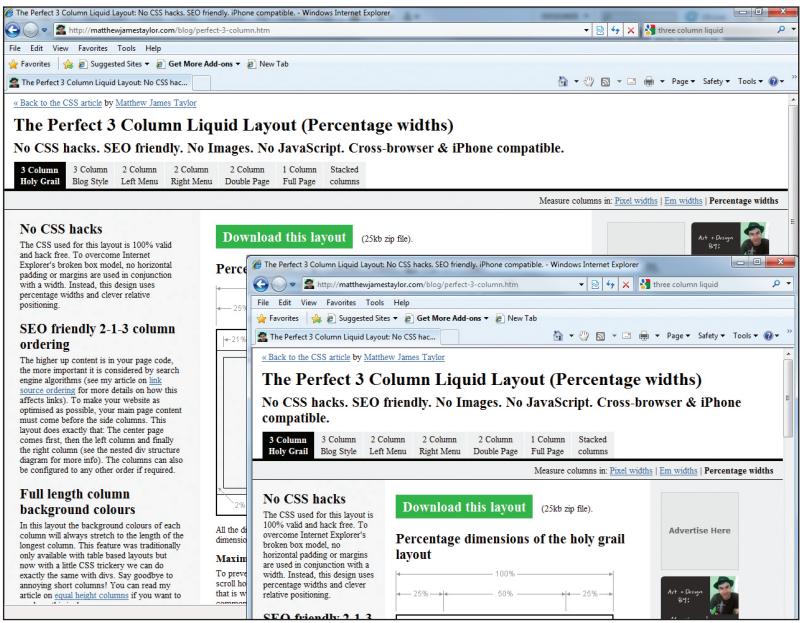
A fixed-width layout has its total width and the widths of its columns defined using static width measurements, typically pixels. A fixed-width layout does not stretch to fill the browser window, and remains its set width. By having a predetermined width for the whole layout and its columns, you can be certain that the window width and screen resolution will not compromise the design, specifically with regard to carefully measured internal elements such as banners, images, and positioned text.

But, this also means that whatever width you declare is served to everyone. A 780-pixel-width design might look great on an 800×600 screen resolution, but it starts to look a bit dwarfed on the more common widescreen laptop screens with a 1280×1024 screen resolution. Also, anyone viewing your site with a viewport of less than 780 pixels is going to get horizontal scrollbars.

9.1.2 Liquid

A liquid (or fluid) layout expands and contracts to fill the browser window. Typically the columns will have widths declared using percentage measurements,

where the browser window (or possibly an all-encompassing containing element) is 100 per cent. An advantage of a liquid layout is that it adapts to suit the available viewing space. However, while this is favourable for window widths from, say, 700 pixels to 1024 pixels, at larger window sizes sentences can become incredibly long and difficult to follow line by line. It is also more difficult to design images and banner elements to fit liquid columns, where graphics may need to acknowledge the stretching or expanding containing element and attempt to fit. The max-width CSS property can be applied to stop the expansion of the layout at a set width. Regretably, IE6 and below on the PC do not support this.



A Liquid Layout

9.1.3 Elastic

The concept of the elastic layout is interesting, but it has its difficulties. While an elastic layout isn't fixed, it is also stopped from getting too wide, as you can specify a maximum width and a minimum width in pixels, ems, or percentages. The benefit here is how the whole layout can scale when text is resized. The elastic layout is a more advanced layout option beyond our scope right now, but if you are curious, Patrick Griffith's article "Elastic

Design" (www.alistapart.com/articles/elastic/) is a good place to start.

9.1.4 Variable fixed width

This sounds like a contradictory term or oxymoron. The idea is that the layout changes automatically to best accommodate the user's window size. For example, if the browser window is wide enough, the layout may contain three fixed-width columns. If the window width falls below a particular width, one column is seamlessly placed under another, creating a two-column layout with the entire markup still present, just reshuffled. The most reliable method is to use JavaScript to assess the window width and change the CSS of the web page as a result. Similar results can be achieved using pure CSS.

9.2 HTML Template

Let us again start by preparing a basic HTML file you can use to follow the examples in this chapter. Create a new file called `columns.html`, containing the following HTML:

```
<html>
<head>
<title>Chapter 11: Classic Layouts</title>
<link rel='stylesheet' media="screen" type='text/css'
href='columns.css' />
</head>
<body
...column content goes here...
</body>
</html>
```

Save the file. Create columns.css and place the following rules into it:

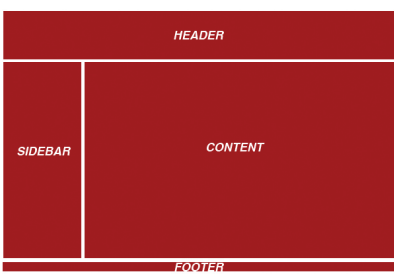
```
/* Specify blanket rules for all elements */
body {
font-size:80%;
font-family:'Lucida Grande',Verdana,sans-serif;
margin:10px;
}
/* Rules for headings */
h1 {
font-size:150%;
}
```

```
h2 {
font-size:140%;
}
h3 {
font-size:120%;
}
/* Default paragraph styles */
p {
font-size:100%;
line-height:150%;
}
```

The style sheet contains nothing special at this stage, just a few simple rules to control the look of the text, and a 10-pixel margin on the body element.

9.3 Liquid floated two-column layout

The goal is to create a simple two-column layout, featuring a main column and a sidebar that stretches to fit the width of the browser window. All examples in this chapter will involve two main sections, masthead and footer. They are almost always required in any layout, especially when first getting to grips with CSS layout. The masthead (also called the header) stretches the full width of the layout and typically holds a logo and possibly the main navigation, plus any other important tools such as a search box or accessibility links (access key information, style switcher, and so on). The footer tidies the whole thing up at the base of the layout, again stretching the full width of any columns. The footer is typically used to store important information such as copyright info, links to legal information (accessibility statement, terms and conditions, etc.).



A Liquid floated two column layout

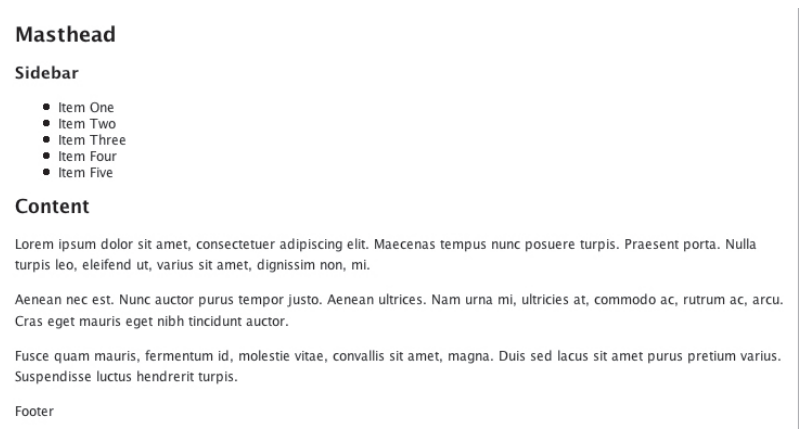
We can now think about the structural HTML needed to divide the page into specific sections. Remember that every time you create a new column , you also create a new parent element for any elements it contains. This means that you can be even more specific with your CSS and how you target certain elements using contextual selectors.

The structural HTML can now be

added to the columns.html file. For this method, it is very important that the sidebar ID appears before the content ID in the markup, to ensure that the top edges of the two columns will line up. This is not entirely semantic in approach, as it is likely that you will want your main content to come first.

```
<body> element of columns.html.
<div id="masthead">
...masthead content goes here...
</div>
<div id="sidebar">
...sidebar content goes here...
</div>
<div id="content">
...main content goes here...
</div>
<div id="footer">
...footer content goes here...
</div>
```

Save the file and load it in the browser. At this stage everything is still totally linear.

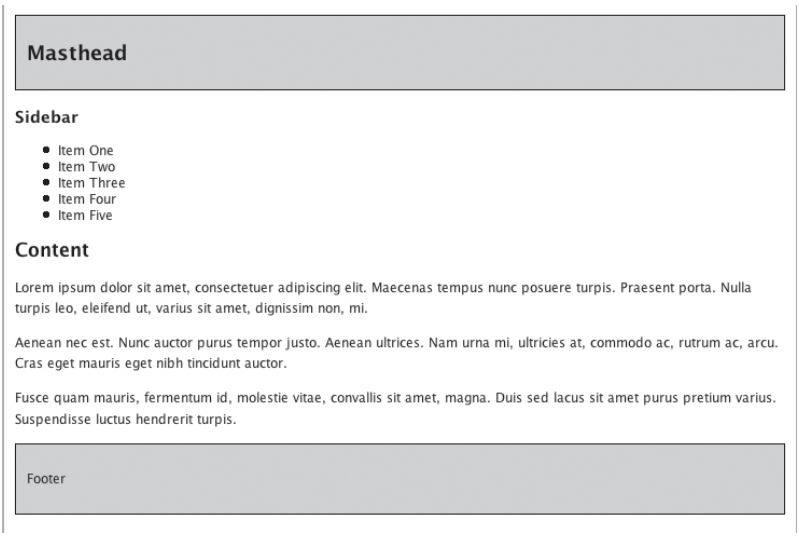


The document is divided into four main sections and the browser defaults style the page

Let's apply some basic styles to two of our sections, specifically masthead and footer. All we're going to do here is add identical rules for each,

defining the section with a black border and gray background, and apply some padding.

```
/* Masthead */
#masthead {
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
/* Footer */
#footer {
padding:10px;
border:1px solid #000;
background-color:#CCC;
```



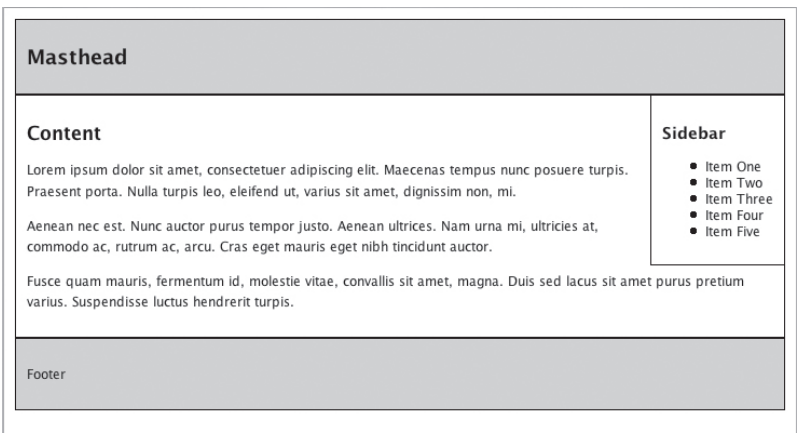
The sections are clearly defined using CSS rules

```
}
```

The first step is to create the illusion of two columns by floating the sidebar in. First, the content ID rules are declared, virtually the same as the rules for masthead and footer except the background color. Identical rules are declared for the sidebar ID, with the crucial addition of float:right, which

will force the sidebar to move into the right.

```
/* Masthead */
#masthead {
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
/* Content */
#content {
padding:10px;
border:1px solid #000;
}
/* Sidebar */
#sidebar {
float:right;
padding:10px;
border:1px solid #000;
}
/* Footer */
#footer {
padding:10px;
```



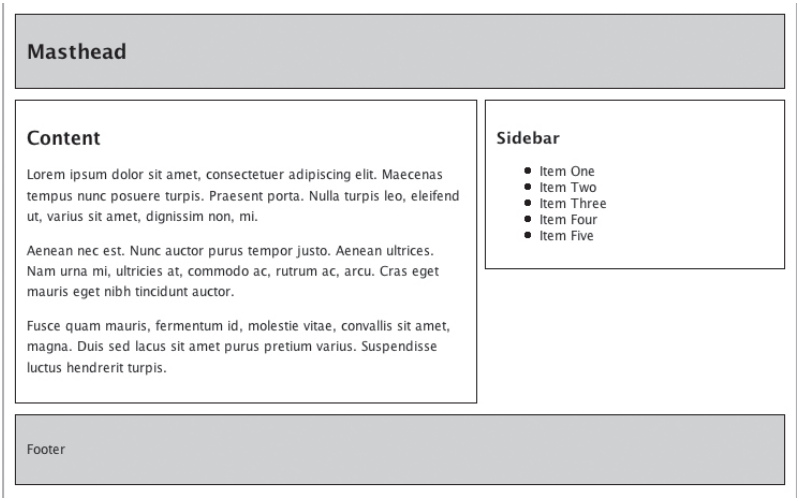
The sidebar is floated to the right

```
border:1px solid #000;
background-color:#CCC;
}
```

Note that the sidebar is only as wide as its content, and that the text contained in the content ID flows neatly around it. The content ID still stretches the full width of the window, and the sidebar is really just a floated box within it. To achieve real columns, we need width and margin. Decision upon the width ratio for your columns should be made. As this layout is liquid, and we are dealing with a layout of 100 per cent width of the browser window, our two columns need to be defined using percentages. A percentage width is declared for the sidebar, the remaining will be for the second column. The next step is to declare a right margin for the content ID that is just a bit larger than the sidebar. If the right margin were the same width as the sidebar, the two sections would budge up against each other, so a slightly larger margin is declared to ensure some space between the two. Here, we shall choose a right margin of 40 per cent. Doing this will mean that the main content area will be 60 per cent wide by default (100 – 40 = 60 per cent). The result is a space to the right of the content ID in which the sidebar can comfortably fit, with 4 per cent to spare. This 4 per cent is the gutter between the two columns. Note as well that a margin value has been declared for masthead, and that bottom margins for content and sidebar have also been added to allow the content and sidebar IDs a little space, and that clear:both has been added to the footer to ensure that it will always appear below the two columns.

```
/* Masthead */
#masthead {
margin:0 0 10px 0;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
/* Content */
#content {
margin-right:40%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
```

```
}
/* Sidebar */
#sidebar {
float:right;
width:36%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```



We declare the margins to create space between the elements

```
/* Footer */
#footer {
clear:both;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
```

The columns have a thin border around them and it is clear that they are of an uneven height. Without the borders it wouldn't matter, but with borders or a background colour, it's going to look weird. We will look at little

trick called "faux columns" a little later, which is a technique that makes use of a simple tiled background image to fool us into thinking that columns are of the same equal height.

The two column layout is more than enough to create a typical blog layout or simple web site. There are, however, many more layout options available to you, including alternative methods of creating a liquid two-column layout, which we'll look at next.

Here each column has been given a bottom margin of 10 pixels to create a space between them and the footer. So, instead of having to add this declaration to every column, can it not just be added to the footer using a top margin. Unfortunately this cannot be done. When you clear a float, the top margin of the clearing element (the footer in this case) is automatically increased so that the top border clears the bottom outer edge of the floated elements. Therefore, with no top margin on the footer and no bottom margins on the columns, the minimum clearance is done automatically, and in this case, 10 pixels is the same more or less as the automatically added top margin, so there is no difference. So you must use a bottom margin on the floated columns in order to create a visible margin between them and the cleared footer.

9.4 Liquid float left, float right

The previous method of floating just the sidebar was good. Still, from a semantic point of view, the main sections were in the wrong order in the markup (sidebar before content). The beauty of the `float:left/ float:right` approach is that we can return to more semantically structured markup.

If you are still working through the examples, first switch the order of the content and sidebar IDs in columns.css, so that the order is as follows:

```
<div id="masthead">
...masthead content goes here...
</div>
<div id="content">
...main content goes here...
</div>
<div id="sidebar">
...sidebar content goes here...
</div>
<div id="footer">
```

```
...footer content goes here...
</div>
```

The CSS code would look like,

```
/* Content */
#content {
float:left;
width:54%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar */
#sidebar {
float:right;
width:36%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

If no padding was set for either ID, the two widths could be larger. You could use, for example, a content ID of 60 per cent width, and a sidebar ID of 36 per cent, which would give a space in between the two of 4 per cent. However, with 10 pixels of padding declared for each (totalling 40 pixels in width), the content ID has a smaller width of 54 per cent to compensate for this.

9.5 Liquid floated three-column layout

Let's jump back to the first floated sidebar example, where to succeed the sidebar element needs to be placed before the content element. Here's the HTML again, but notice that the sidebar ID has now been removed, and instead we have two new sidebars, one called `sidebar_a`, and one called `sidebar_b`, just above it.

```
<div id="masthead">
...masthead content goes here...
</div>
<div id="sidebar_a">
```

```
...sidebar content goes here...
</div>
<div id="sidebar_b">
...sidebar b content goes here...
</div>
<div id="content">
...main content goes here...
</div>
<div id="footer">
...footer content goes here...
</div>
```

Next the CSS used for the floated sidebar method. As there will be an extra column in play pretty soon, it is sensible to decrease the width of the existing sidebar to 25 per cent, and allow less space for it (30 per cent) in the content ID. Importantly, the sidebar ID used earlier has gone, so a new selector has been added called sidebar_b to match one of the new HTML elements.

```
/* Masthead */
#masthead {
margin:0 0 10px 0;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
/* Content */
#content {
margin-right:30%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar B */
#sidebar_b {
float:right;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
```

```
}
/* Footer */
#footer {
clear:both;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
```

Notice that as the `sidebar_a` element appears first in the markup, and as of yet has no matching selector, it floats above the reworked columns. The next stage is to create a space to the left of the content ID in which the new sidebar can be floated. This calls for a left margin declaration for the content selector.

```
/* Content */
#content {
margin-left:30%;
margin-right:30%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

Finally, rules for the new sidebar_a ID are declared. The values are identical to those of the existing sidebar_b, except that in this instance sidebar_a is floated left.

```
/* Sidebar A */
#sidebar_a {
float:left;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

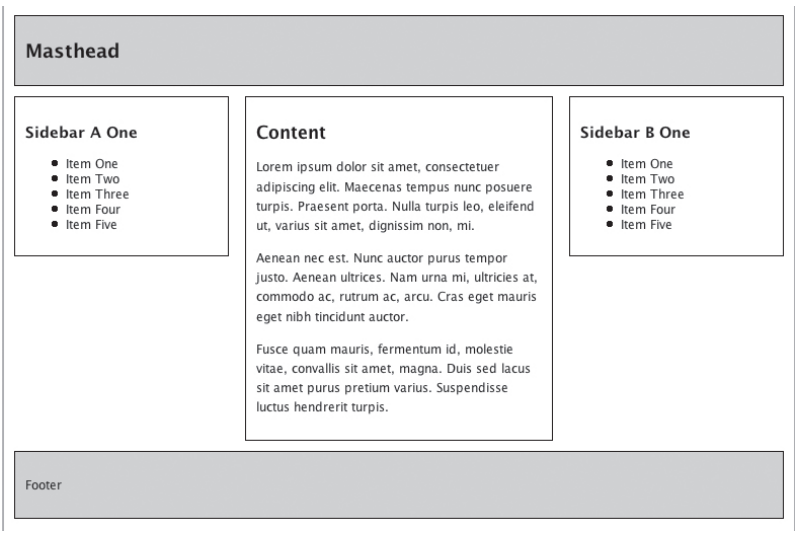
Now that one sidebar is being floated left and one floated right, and owing to the adequate left and right margins of the content ID, all three columns are sitting perfectly, and will stretch and contract to fit the browser window without collapsing. You are about to witness the true power of CSS layout. Notice that in each sidebar, the list headings contain either an A or a B,

depending upon whether they are in sidebar_a or sidebar_b. Remember that the two selectors are as follows:

```
/* Sidebar A */
#sidebar_a {
float:left
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar B */
#sidebar_b {
float:right;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

To switch the position of the columns from left to right and vice versa, just one simple change to each selector is required. Basically, just switch the two rules over.

```
/* Sidebar A */
```



```
#sidebar_a {
float:right;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar B */
#sidebar_b {
float:left;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

Simply by swapping the float values over, the columns are moved. This trickery is made possible due to the symmetrical nature of this layout. Both sidebars are the same width and have almost identical properties, and the space allotted to each by the content ID's margins is identical.

9.6 Liquid positioned two-column layout

You can stick with the markup used in the previous floated two-column layout example (where content precedes sidebar). Using positioning for layout can be advantageous as there is no correlation between the order of the sections in the markup and their final positions when styled with CSS positioning. This time, however, there will be changes to the masthead CSS.

In this example, a set height is required for the masthead, as this will be used as a reference to help us position the sidebar in a little while. This set height would typically be informed by the content of the masthead. Let's imagine that a company logo is to be placed in the masthead and that it is 60 pixels in height. This measurement informs the first step toward a positioned layout.

```
/* Masthead */
#masthead {
height:60px;
margin:0 0 10px 0;
border:1px solid #000;
background-color:#CCC;
```

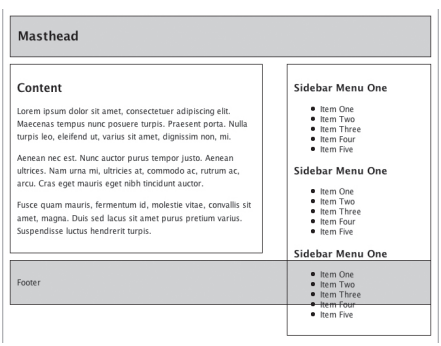
```
}

/* Apply padding to heading to avoid Box Model woes */
h1 {
padding:0 0 0 10px;
}
```

The padding has also been removed from the masthead at this stage to keep things simple, and instead added to the h1 selector. (To find out why, skip to the section "The Box Model" later in this chapter.) The next step is to declare a right margin for the content ID as we did in the floated sidebar method earlier in the chapter. When the sidebar is positioned later, it will be slotted into the space created by this right margin.

```
/* Content */
#content {
margin-right:40%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

As mentioned, the sidebar now needs to be positioned into the space to the right of the content ID. It is very important when using positioning to be mindful of any extra issues that might flaw your positioning. In this example, note that there is 10 pixels of padding around the whole body element, and that there is also a 10-pixel margin directly below the masthead. There is also a 1-pixel border around each edge. We need to add these measurements to the height of the masthead (60 pixels) in order to know exactly how far from the top of the browser window the sidebar needs to be. In this case, it's 10 + 1 + 60 + 1 + 10, equaling 82 pixels from the top of the window.



The sidebar overlaps the footer if it has more elements

```
/* Sidebar */
#sidebar {
position:absolute;
```

```
top:82px;
right:10px;
width:30%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

Note that right:10px is also specified to ensure that the sidebar does not push up against the right edge of the browser window, and therefore honours the 10 pixels of padding around the <body> element.

This approach has a hidden problem. Once the sidebar contains more elements the sidebar overlaps the footer. Remember that with a floated sidebar, the footer could be cleared to ensure it began after the columns. With positioning, however, the sidebar has been taken out of the normal flow of elements, and the footer can't "see" this. The best approach here is to make the footer work like the content ID, by giving it a right margin and thus creating space for the sidebar.

```
/* Footer */
#footer {
margin-right:40%;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
```

The footer no longer stretches the full width of the layout. Another problem is that the greater the difference in amount of content between content ID and sidebar ID, the more unbalanced the layout will look.

9.7 Liquid positioned three-column layout

Moving from two columns to three using the positioning method isn't too difficult. Remember how we adjusted the right margin of the main content ID to allow space for a sidebar? Well, all we have to do is something similar with the left margin. First of all, sidebar is removed, and two new sidebars need to be added to the markup. They can be placed anywhere, seeing as CSS positioning will be used to place them visually. To stay semantic, let's add them after the content ID. Again, these changes can be made to the `columns.html` file.

```
<div id="masthead">
...masthead content goes here...
</div>
<div id="content">
...main content goes here...
</div>
<div id="sidebar_a">
...main content goes here...
</div>
<div id="sidebar_b">
...sidebar content goes here...
</div>
<div id="footer">
...footer content goes here...
</div>
```

If you are working through this example, be sure to amend the CSS selector if you have changed the name of your sidebar. Now, let's make some space to the left of the content ID by declaring a left margin wide enough to accommodate the new sidebar, plus a little space between the two.

```
/* Content */
#content {
margin-left:30%;
margin-right:30%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar A */
#sidebar_a {
position:absolute;
top:82px;
left:10px;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
/* Sidebar B */
```

```
#sidebar_b {
position:absolute;
top:82px;
right:10px;
width:25%;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

For each sidebar, the widths are a little smaller than in previous examples, as we are of course allowing less space for each sidebar. Finally, to prevent sidebar_a from overlapping the footer if there is too much content within it, a left margin identical to that of the content ID needs to be declared for the footer.

```
/* Footer */
#footer {
margin:0 30% 0 30%;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
```

These result in a perfect positioned three-column layout that stretches to fit the width of the browser window.

9.8 Fixed-Width Layout

Previous layouts in this chapter have been liquid in their approach, with widths declared using percentage values. It is of course perfectly possible and very common, to use set widths declared using pixels. This approach can provide greater control and is still favoured by many web designers. With the troublesome browsers in mind, let's look at an absolute fundamental of CSS layout—the Box Model.

9.9 The Box Model

Before you start using fixed widths for your columns, it is imperative that you get to grips with the Box Model. If you think you might need to apply margins or padding to any of your columns, you need to be aware of the miscalculations these will cause in IE5 and IE5.5 on a PC.

In any standards-compliant browser, the total width for a containing

element and its padding and border is calculated as the combined values of that container's width plus its padding and border. This means that a 300px container is 300px plus values for padding and border. This is how it should be. However, IE5 and IE5.5 get this wrong by subtracting the widths of the border and padding from the width value. This means that a 300px container ends up being much narrower. If padding is declared as 20px, then the actual width of the container in these browsers will be 300 – 20 – 20, equaling 260 pixels.

Let's examine this using the following simple declaration as an example:

```
/* Sidebar_a */
#sidebar_a {
width:300px;
padding:10px;
border:10px solid #000;
}
```

Although the sidebar itself is 300 pixels in width, the space required to accommodate it needs to be equal or greater than all the widths added together. This will be border-left + padding-left + width + padding-right + border-right, which equates to 10 + 10 + 300 + 10 + 10, giving a total width of 340 pixels.

With the Box Model under your belt, you can fix your widths with confidence. For this example, we'll take the liquid floated three-column layout created earlier. Here's the HTML again:

```
<div id="masthead">
...masthead content goes here...
</div>
<div id="sidebar_a">
...sidebar content goes here...
</div>
<div id="sidebar_b">
...sidebar content goes here...
</div>
<div id="content">
...main content goes here...
</div>
<div id="footer">
...footer content goes here...
</div>
```

Next the existing CSS from the liquid floated three-column layout can be reworked for our fixed version. The first decision to make is how wide the layout should be.

For this example, the total width for the layout would be 760 pixels. The first step is to define this width in the body declaration.

```
/* Specify blanket rules for all elements */
body {
width:760px;
margin:10px;
font-size:80%;
font-family:'Lucida Grande',Verdana,sans-serif;
}
```

Doing this will ensure that any element floated right will not seek to stick to the right edge of the browser window.

With the figure of 760 pixels in mind, the properties of both masthead and footer need examining. Paying consideration to the Box Model, if both masthead and footer are to be 760 pixels in width, their specified width needs to be 760 pixels minus border and padding. That gives us 760 – 1 – 10 – 10 – 1 = 738. Therefore the declared widths will be 738 pixels.

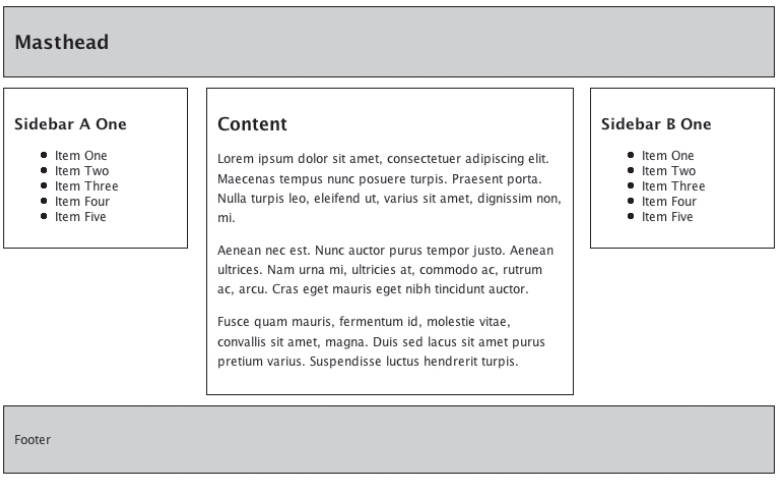
```
/* Masthead */
#masthead {
width:738px;
margin:0 0 10px 0;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
/* Footer */
#footer {
clear:both;
width:738px;
padding:10px;
border:1px solid #000;
background-color:#CCC;
}
```

Now the columns need to have their set widths declared.

Here, the final width value of the content ID is somewhat dictated by the fact that we need to allow for existing border, padding, and margins. Taking

values from left to right we have 200 + 1 + 10 + 338 + 10 + 1 + 200, equaling 760 pixels.

```
/* Content */
#content {
width:338px;
margin-left:200px;
margin-right:200px;
margin-bottom:10px;
```



Final complete three column layout

```
padding:10px;
border:1px solid #000;
}
```

From this, we know that each sidebar has a maximum allowance of 180 pixels. The margins suggest 200 pixels are allowed, but this includes an extra 20 pixels either side of the content ID to space the columns apart. Here we need to get really mathematical. Each sidebar must be no wider than 180 pixels, including padding and border values. Subtracting the border and padding values from 180 gives us a width value of 158 pixels for each sidebar (180 – 1 – 10 – 1 – 10 = 158).

```
/* Sidebar A */
#sidebar_a {
```

```
float:left;
width:158px;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}

/* Sidebar B */
#sidebar_b {
float:right;
width:158px;
margin-bottom:10px;
padding:10px;
border:1px solid #000;
}
```

Now we can safely rest in the knowledge that the combined widths of both sidebars and main content do not exceed 760 pixels, we can be sure of the layout.

10 Tips & Tricks

This chapter aims to draw together the most common tips, tricks, and troubles. Over the next few pages, you'll find ideas for maximizing page space by manipulating overflowing content and learn to create incredibly simple CSS rollover images.

10.1 Rollover Images

A rollover image adds a simple bit of interaction for the user. Place the cursor over a rollover image, and another replaces it. For years, web designers preloaded on-state (i.e., on mouseover when the mouse is rolled over the image) images to assist the browser in its rollover presentation.

Preloading increases the weight of the initial download, but adds to usability by decreasing the wait time for an on-state image to appear. Basically, the browser would only start downloading the on-state image upon rollover..

All that is required in the HTML is the addition of a class to the <a> element.

```
<a href="target.html" class="rollover">Big Lee Hickman</a>
```

This unobtrusive additional markup allows you to take control of this particular link in any way you see fit using your style sheet.

Just one image is required for this technique. Both image states are combined as one image. This means that when the user rolls over the image, the on-state image is already available. The trick is to place the two versions side by side as one image, making very sure that each half is of an identical size. With this dual-purpose image prepared, we can now think about the CSS that will be used to slide the image backward and forward depending on the link state.

First, the rollover class is declared. All that is required is to assign the width and height of the visible area of the dual image and set the display to block (to ensure the dimensions are respected). The image itself is added using the background property, set to no-repeat.

```
/* Rollover class */
.rollover {
display:block;
width:210px;
height:157px;
background:url(/images/rollover.jpg) no-repeat;
```

```
text-indent:-9999px;
}
```

Note also that the text is indented using a huge negative value (text-indent:-9999px). This ensures that the link text does not show, unless the style sheet is turned off or unavailable. In that situation, the user would not see the image, but would still have a clickable text link.

In the previous chapters you learned about the various pseudo link states. Well, these will now be used to create the rollover effect. The three states :link, :visited, and :active are grouped, as they share the same values. For each of these states, the first half of the dual image will be visible.

```
a.rollover:link, a.rollover:visited, a.rollover:active {
background:url(/images/rollover.jpg) no-repeat;
}
```

The most important step is to use the :hover state to reposition the dual image so that the second half (the rollover state) is in view. To do this, the image is repositioned 210 pixels to the left using a negative position value.

```
a.rollover:hover {
background-position:-210px 0;
}
```

This simple method slides the first half of the dual image out of view, bringing the second half into the available space. This happens instantly upon rollover, creating a seamless, simple rollover effect. It should go without saying that this technique is invaluable for all kinds of interactive images, especially buttons and navigation items. Just make sure that the link still makes sense should the dual image be unavailable.

10.2 The overflow property

The overflow property defines the way that a child element is displayed when it exceeds its containing element. In other words, if there is too much content, the overflow value will dictate how or whether it should be displayed. There are four possible values for the overflow property. Let's look at two values in particular: auto and hidden.

10.2.1 overflow:auto

overflow:auto trickery (whereby specifying the height of a and applying

overflow:auto creates a mock <iframe> without all the accessibility headaches of <iframe>s). For this job, all pages needed to be the same height all the way across, regardless of how much content was used.
Visible - The content is not clipped. It renders outside the element.
Hidden - The scrollbar is not displayed to see the rest of the content.
scroll - The scrollbar is displayed even if there is nothing to scroll.
auto - The scrollbar is displayed depending on the content

This was achieved by using a container with overflow:auto applied and a set height, ensuring that no matter how much information appears in the container, the total height of the page does not increase. Let's look at creating our own example—a simple news stream using CSS.

All that is needed is a simple container. Here, it is given a class of stream to denote that it carries all the streamed news items.

```
<div class="stream">
<p>All the stuff you want to scroll goes in here.</p>
</div>
```

Note that for this example, you will need to paste plenty of content into this container to see the effect of the overflow.. With that taken care of, we can move on to the nifty styling. Now define the stream selector. First, the size of the box is declared (here it is a square 300×300 pixels), and some padding is declared to ensure the content does not touch the border. Most importantly, the overflow property is given the auto value.

```
.stream {
width:300px;
height:300px;
padding:10px;
border:1px solid #999;
background-color:#FFF;
overflow:auto;
}
```

By giving the box a set height, you ensure that it does not expand or contract based on the content it holds. If there is more content than the box can hold, the scrollbar will automatically appear, allowing you to scroll through the whole text.

The possibilities here are endless, but be wary of using multiple overflow boxes on your pages. A web page that features umpteen scrollbars can confuse users though.

10.2.2 overflow:hidden

There are occasions when you don't want the overflowing content to show, or you only wish it to appear in certain circumstances. This is where overflow:hidden comes in handy. For example, you might have a beautiful photograph of some mountains that you wish to use as a banner image. This image is 796 pixels wide, and you know that if you add it to the page, it will always force the site to be at least that wide, and would force horizontal scrollbars if the window was decreased below that width.

For this example, we will use an image that is 796 pixels in width by 320 pixels in height. The image needs to be placed into the (X)HTML page as follows:

```
<div id="masthead"><a href="#"><img src="images/mountains.jpg" alt="Mountain banner" width="796" height="320" /></a></div>
```

You can see the context into which the banner will be placed. Following that, the CSS for the masthead ID is shown. Note that the width is a percentage value (100%) ensuring the masthead always fills the available space. The height is equal to that of the image it will hold. Finally, overflow:hidden is specified.

```
/* Container for all page content */
#container {
border:1px solid #000;
padding:20px;
background-color:#FFF;
}
/* Masthead */
#masthead {
width:100%;
height:320px;
border:2px solid #999;
background:#CCC;
overflow:hidden;
}
```

That is all you need to do. In the browser under normal circumstances, the thinnest window would force a horizontal scrollbar due to the larger width of the image. Thanks to hiding the overflow, this is skilfully avoided.

10.3 Combining classes

It is possible to combine classes. This functionality provides real power when it comes to reusing elements. For example, you can use a containing element as often as you want, but you may not always want the elements it contains to be displayed in the same way. So, you could make several versions of the containing element and set the unique properties for each, but why would you want to repeat the margin, padding, and background styles for each and end up with more class names to worry about? This is where combined classes are really useful. Now, if you wanted an <h3> inside a particular container to have a red background, you might create the following styles:

```
h3 {
font-size:110%;
margin:10px 5px 10px 5px;
padding:5px;
}
.custom_background {
background-color:#F00;
}
```

You might then call that as follows:

```
<div class="container">
<h2>Latest News</h2>
<h3 class="custom_background">Man found in vase</h3>
content
</div>
```

No! This is extra markup forced to use a class attribute with each heading to define its colour. Instead, create unique selectors for each section, keeping the existing <h3> declaration, and then creating a set of background styles with semantically meaningful names:

```
.news h3{
background:#F00;
}
.entertainment h3{
background:#666;
}
.sport h3{
font-size:110%;
background:#CCC;
```

```
}
.music h3{
background:#999;
}
```

Then, all you need to decide is the purpose for each container. Let's say you add a container to house articles about sport. Add the sport attribute to the existing container class, separating the two class names with a whitespace character:

```
<div class="container sport">
<h2>Sports News</h2>
<h3>England win 26-0</h3>
content
</div>
```

Suddenly, all instances of a <h3> heading inside this container take on the light gray background colour. Exchange the word "sport" for "music" and now all headings in that container have a darker gray background. Remove the heading rule entirely, and headings have no background colour at all.

Now that you know all the basic commands and properties in CSS and how you can work with them, you can experiment and build your own techniques. The art of CSS can only be mastered by practice. Learning the concepts is a simple task, but you can create truely creative compelling content only once you have gained a lot of experience. If you like a site design, have a look at its source and check out its CSS. All the best. d

Notes